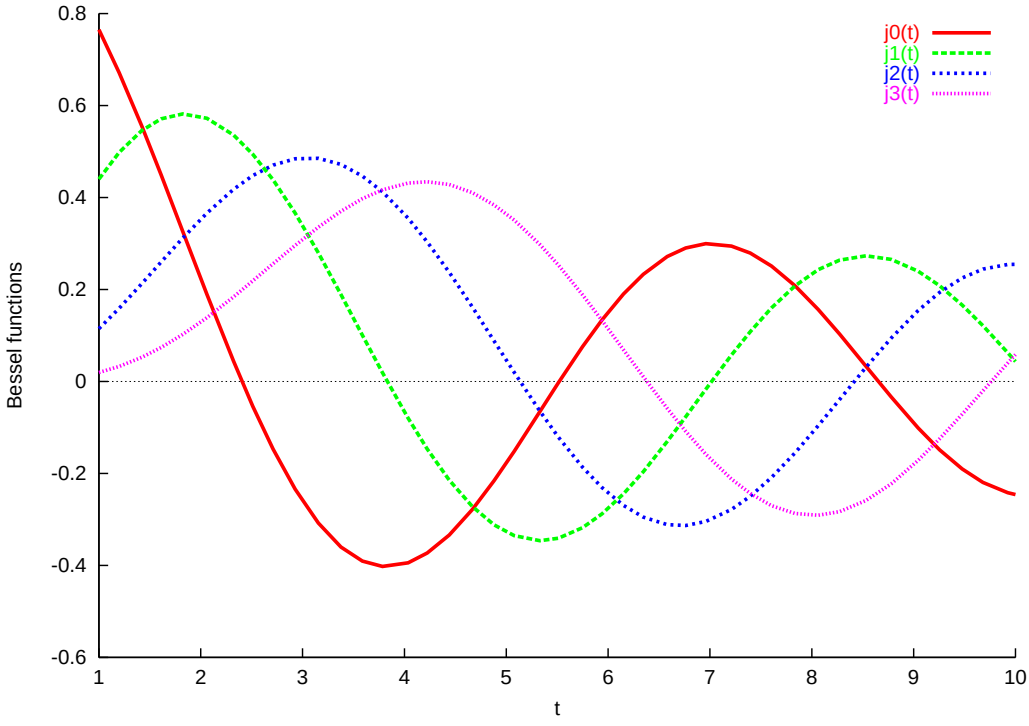
SoftIntegration®

Ch 言語環境

バージョン 6.3

リファレンス ガイド



SoftIntegration 社の連絡先

住所	SoftIntegration, Inc.
	216 F Street, #68
	Davis, CA 95616
電話	+ 1 530 297 7398
Fax	+ 1 530 297 7392
Web	http://www.softintegration.com
メール	info@softintegration.com

2010 年 2 月 日本語版 6.3

まえがき

Ch は C 互換のクロスプラットフォームのスクリプト言語環境です。Ch はまた、C の仮想マシンであり、C++、他の言語およびソフトウェアパッケージからの顕著な機能をともなった、C インタープリタのスーパーセットです。Ch の関数とクラスの説明は、この『リファレンス ガイド』で提供されます。Ch 言語環境の整理に関するヘッダファイルの一覧は、表 1に与えられています。このリファレンスマニュアルで説明されていない関数[1]については、ISO C および POSIX standard のような、ISO および現存する標準規格に従います。

表記規則

以下のリストは、このドキュメント全体にわたる、テキストの特定の要素に対する表示形式として使用されている、表記規則について定義し、説明しています。

- インターフェース コンポーネント は、モニタ画面または表示上に現れるウィンドウ タイトル、ボタン、アイコン名、メニュー名、選択項目、およびその他のオプションです。これらは、太字で表されています。マウスでの一連のポイントとクリックは、一連の太字の単語で表されています。

  例: **OK** をクリックします

  例: シーケンス **[スタート]**->**[プログラム]**->**[Ch6.3]**->**[Ch]** は、最初に **[スタート]** を選択し、次にマウスで **[プログラム]** をポイントし、**[プログラム]** サブメニューを選択して、**[Ch6.3]** を選び、最後に **[Ch]** を選択することを示しています。

- キーボードのキー上にあるラベル、キーキャップ は、山かっこで囲まれています。キーキャップのラベルは、タイプライターのような書体で表されています。

  例: `<Enter>` を押します

- キー組み合わせ とは、(特に指定がない場合) 一つの機能を実行するために、一連のキーが同時に押されることです。キーのラベルは、タイプライターのような書体で表されています。

  例: `<Ctrl><Alt><Enter>` を押します

- コマンド は、小文字の太字で表されていて、参照のためのみにあり、議論の特定のポイントで意図して入力するものではないことを示しています。

  例: "... をインストールするには、**[インストール]** コマンドを使用します。"

[1]訳注：この日本語版『リファレンス ガイド』では、「2D および 3D プロット」、「複素関数」、「数値解析」、「拡張マルチバイトおよびワイド文字関数」「ワイド文字関数」の各章の関数リファレンスについては日本語訳されていません。

表 1: ライブラリのまとめ

ヘッダファイル	説明	C	POSIX	Ch
aio.h	非同期入出力		X	X
arpa/inet.h	インターネット操作			X
array.h	計算配列			X
assert.h	診断	X	X	X
chplot.h	2D および 3D プロット			X
chshell.h	Ch シェル関数			X
complex.h	複素数	X		X
cpio.h	Cpio アーカイブ値			X
crypt.h	暗号化関数			X
ctype.h	文字操作	X	X	X
dirent.h	ディレクトリエントリのフォーマット		X	X
dlfcn.h	動的にロードされる関数			X
errno.h	エラー番号	X	X	X
fcntl.h	プロセス間通信関数		X	X
fenv.h	浮動小数点環境	X		X
float.h	プラットフォーム依存の浮動小数点の限界値	X	X	X
glob.h	パス名のパターンマッチ型			X
grp.h	group 構造体		X	X
inttypes.h	固定長整数型	X		X
iostream.h	C++スタイルの入出力ストリーム			X
iso646.h	代替表現	X		X
libintl.h	国際化のためのメッセージカタログ			X
limits.h	プラットフォーム依存の整数の限界値	X	X	X
locale.h	ロケール関数	X	X	X
malloc.h	動的メモリ管理関数			X
math.h	数学関数	X	X	X
mqueue.h	メッセージキュー		X	X
netconfig.h	ネットワーク構築データベース			X
netdb.h	ネットワークデータベース操作			X
netdir.h	転送プロトコル用名前－アドレスマッピング			X
netinet/in.h	インターネットプロトコルファミリ			X
new.h	C++スタイルのメモリ割り当てエラーハンドリング			X
numeric.h	数値解析			X
poll.h	poll() 関数用の定義			X
pthread.h	スレッド		X	
pwd.h	パスワード構造体		X	X
re_comp.h	re_comp() 用正規表現マッチング関数			X
readline.h	readline 関数			X
regex.h	正規表現マッチング型			X
sched.h	実行スケジューリング		X	X

表 1: ライブラリのまとめ (続き)

ヘッダファイル	説明	C	POSIX	Ch
semaphore.h	semaphore 関数		X	X
setjmp.h	非ローカルジャンプ	X	X	X
signal.h	シグナルハンドリング	X	X	X
stdarg.h	変数引数リスト	X	X	X
stdbool.h	ブール数	X		X
stddef.h	その他の関数とマクロ	X	X	X
stdint.h	整数型	X	X	X
stdio.h	入出力	X	X	X
stdlib.h	ユーティリティ関数	X	X	X
string.h	文字列関数	X	X	X
stropts.h	ストリームインターフェース			X
sys/acct.h	プロセスアカウンティング			X
sys/fcntl.h	制御ファイル			X
sys/file.h	ファイル構造体配列へのアクセス			X
sys/ioctl.h	制御デバイス			X
sys/ipc.h	プロセス間通信アクセス構造体			X
sys/lock.h	ロッキングプロセス			X
sys/mman.h	メモリ管理宣言		X	X
sys/msg.h	メッセージキュー構造体			X
sys/procset.h	set プロセス			X
sys/resource.h	XSI リソース操作			X
sys/sem.h	セマフォア機能			X
sys/shm.h	共有メモリ機能			X
sys/socket.h	インターネットプロトコルファミリ			X
sys/stat.h	ファイル構造体関数		X	X
sys/time.h	time 型			X
sys/times.h	time 構造体へのファイルアクセスと変更		X	X
sys/types.h	データ型		X	X
sys/uio.h	ベクタ I/O 操作			X
sys/un.h	Unix ドメインソケット			X
sys/utsname.h	システム名構造体		X	X
sys/wait.h	exit ステータスの評価		X	X
syslog.h	システムエラーログ			X
tar.h	拡張 tar 定義			X
termios.h	termios 用の値の定義		X	X
tgmath.h	型に依存しない数学関数	X		X
time.h	時間と日付関数	X	X	X
tiuser.h	トランスポート層インターフェース			X
unistd.h	システムおよびプロセス関数		X	X
utime.h	time 構造体へのアクセスと変更		X	X

表 1: ライブラリのまとめ (続き)

ヘッダファイル	説明	C	POSIX	Ch
wait.h	停止または終了のための子プロセス待ち			X
wchar.h	複数バイトの I/O および文字列関数	X		X
wctype.h	複数バイト文字クラスのテスト	X		X

注意: シンボル 'X' は、このライブラリが標準でサポートされていることを示しています。

これに対して、タイプライターのような書体で表されたコマンドは、命令の一部として意図的に入力されます。

例: "ソフトウェアを現在のディレクトリ上にインストールするには、`[install]` を入力します。"

- コマンド書式行 は、コマンドとそのすべてのパラメータから構成されます。コマンドは小文字の太字で表示され、(ある値で置き換える) 変数パラメータは、小文字のイタリック体で表示され、定数パラメータは小文字の太字で表示されます。かっこは、オプションの項目を示しています。

  例: **ls** [**-aAbcCdfFgilLmnopqrRstux1**] [*file* ...]

- コマンド行 は、コマンドと場合によっては、いくつかのコマンド パラメータで構成されます。コマンド行は、タイプライターのような書体で表されます。

  例: `ls /home/username`

- 画面テキスト は、画面または外部モニタ上に表示されます。これはたとえば、システム メッセージ、または (コマンド行のように参照される) コマンドの一部として入力する可能性があるテキストです。画面テキストは、タイプライターのような書体で表されます。

  例: 以下のメッセージが画面上に表示されます

  ```
  usage:  rm [-fiRr] file ...


  ls [-aAbcCdfFgilLmnopqrRstux1] [file ... ]
  ```

- 関数プロトタイプ は、返値、関数名、およびデータ型とパラメータをもった引数で構成されます。Ch 言語のキーワード、typedef 名、および関数名は太字体で表されます。関数引数のパラメータは、イタリック体で表されます。かっこは、オプション項目を示しています。

  例: **double derivative**(**double** (**func*)(**double**), **double** *x*, ... [**double** **err*, **double** *h*]);

- プログラムの ソース コード は、タイプライターのような書体で表されます。

  例: 以下のコードのプログラム **hello.ch**

```
int main() {
    printf("Hello, world!\n");
}
```

は、画面上に、 `Hello, world!` を表示します。

- 変数 は、ある値で置き換えるべきシンボルです。これらはイタリック体で表されます。

  例: module *n* (ここで、*n* はメモリ モジュール番号を表します)

- システム変数とシステム ファイル名 は、太字体で表されます。

  例: Unix 上のスタートアップ ファイル **/home/username/.chrc** または `/home/username` ディレクトリの **.chrc** および、Windows 上の **C:_chrc** または C:\ ディレクトリの **_chrc**

- プログラム内で宣言された 識別子 は、テキスト内で使用される場合、タイプライターのような書体で表されます。

  例: 変数 `var` がプログラム内で宣言されます。

- ディレクトリ は、テキスト内で使用される場合、タイプライターのような書体で表されます。

  例: Ch は、Unix ではディレクトリ `/usr/local/ch` に、Windows ではディレクトリ `C:\Ch` にインストールされます。

- 環境変数 は、システム レベルの変数です。これらは太字体で表されます。

  例: 環境変数 **PATH** はディレクトリ /usr/ch を含みます。

他の関連ドキュメント

Ch のマニュアル構成は、以下のとおりとなっています。これらのマニュアル（PDF 形式）は、製品 CD に同梱され、CHHOME/docs （ここで CHHOME は Ch のホーム ディレクトリ）にインストールされます。

- *The Ch Language Environment — Installation and System Administration Guide*, version 6.3, SoftIntegration, Inc., 2009.

  このマニュアルは、ウェブサーバ用の Ch のスタートアップ方法だけでなく、システムインストールと構築について述べています。
  （注）このマニュアルは日本語化されていません。製品のインストール方法（Windows 版）については、製品パッケージ同梱の『補足説明書』を参照してください。

- *Ch* **言語環境、— ユーザーズ ガイド**, version 6.3, SoftIntegration, Inc., 2009.

  このマニュアルは、様ざまなアプリケーションに対する Ch の言語機能について記載しています。

- *Ch* **言語環境、— リファレンス ガイド**, version 6.3, SoftIntegration, Inc., 2009.

  このマニュアル（本書です）は、サンプル ソースコードを伴った、関数、クラスおよびコマンドの詳細なリファレンス情報を提供しています。現在のところ日本語版では、「2D および 3D プロット」、「複素関数」、「数値解析」、「拡張マルチバイトおよびワイド文字関数」「ワイド文字関数」の各章の関数リファレンスについては日本語訳されていません。

- *The Ch Language Environment, — SDK User's Guide*, version 6.3, SoftIntegration, Inc., 2009.

  このマニュアルは、静的または動的ライブラリにおいて C/C++関数とのインターフェース用のソフトウェア開発キットについてのべています。
  （注）このマニュアルは日本語化されていません。

- *The Ch Language Environment CGI Toolkit User's Guide*, version 3.5, SoftIntegration, Inc., 2003.

  このマニュアルは、CGI クラスにおける Common Gateway Interface と、そのクラスの各メンバ関数に対する詳細なリファレンス情報を提供します。
  （注）このマニュアルは日本語化されていません。

目次

まえがき **iv**

第 1 章　診断テスト — <assert.h> **1**

assert 2

第 2 章　2D および 3D プロット <chplot.h> **4**

CPlot Class 4

arrow 11
autoScale 15
axis 17
axisRange 18
axes 22
barSize 24
border 25
borderOffsets 27
boxBorder 28
boxFill 29
boxWidth 32
changeViewAngle 33
circle 35
colorBox 37
contourLabel 39
contourLevels 41
contourMode 44
coordSystem 46
data 51
data2D 54
data2DCurve 59
data3D 60
data3DCurve 68
data3DSurface 70
dataFile 73
dataSetNum 77
deleteData 78
deletePlots 78

dimension . . . 79
displayTime . . . 80
enhanceText . . . 81
func2D . . . 86
func3D . . . 88
funcp2D . . . 89
funcp3D . . . 91
getLabel . . . 92
getOutputType . . . 93
getSubplot . . . 94
getTitle . . . 98
grid . . . 99
isUsed . . . 102
label . . . 103
legend . . . 104
legendLocation . . . 106
legendOption . . . 107
line . . . 111
lineType . . . 113
margins . . . 117
origin . . . 118
outputType . . . 119
plotType . . . 130
plotting . . . 159
point . . . 159
pointType . . . 161
polarPlot . . . 164
polygon . . . 166
rectangle . . . 170
removeHiddenLine . . . 172
scaleType . . . 174
showMesh . . . 176
size . . . 178
size3D . . . 180
sizeOutput . . . 181
sizeRatio . . . 181
smooth . . . 183
subplot . . . 185
text . . . 186
tics . . . 187
ticsDay . . . 188
ticsDirection . . . 190

ticsFormat . . . 191
ticsLabel . . . 193
ticsLevel . . . 195
ticsLocation . . . 197
ticsMirror . . . 199
ticsMonth . . . 201
ticsPosition . . . 202
ticsRange . . . 203
title . . . 204
fplotxy . . . 206
fplotxyz . . . 208
plotxy . . . 211
plotxyf . . . 217
plotxyz . . . 219
plotxyzf . . . 224

第 3 章　シェル関数 — <chshell.h> **227**
chinfo . . . 229
iskey . . . 231
isstudent . . . 233
isvar . . . 234
sizeofelement . . . 235

第 4 章　複素関数 — <complex.h> **237**
cabs . . . 239
cacos . . . 240
cacosh . . . 241
carg . . . 242
casin . . . 243
casinh . . . 244
catan . . . 245
catanh . . . 246
ccos . . . 247
ccosh . . . 248
cexp . . . 249
cimag . . . 250
clog . . . 251
conj . . . 252
cpow . . . 253
creal . . . 254
csin . . . 255
csinh . . . 256
csqrt . . . 257

ctan . . . 258
ctanh . . . 259
iscnan . . . 260

第 5 章　文字の処理 — <ctype.h> **261**
isalnum . . . 262
isalpha . . . 263
iscntrl . . . 265
isdigit . . . 266
isgraph . . . 267
islower . . . 268
isprint . . . 270
ispunct . . . 271
isspace . . . 272
isupper . . . 273
isxdigit . . . 274
tolower . . . 275
toupper . . . 276

第 6 章　エラー — <errno.h> **277**

第 7 章　浮動小数点型の特性 <float.h> **278**

第 8 章　整数型のサイズ — <limits.h> **281**

第 9 章　ローカライズ <locale.h> **283**
localeconv . . . 286
setlocale . . . 292

第 10 章　算術 <math.h> **294**
acos . . . 298
acosh . . . 299
asin . . . 300
asinh . . . 301
atan . . . 302
atan2 . . . 303
atanh . . . 305
cbrt . . . 306
ceil . . . 307
copysign . . . 308
cos . . . 309
cosh . . . 310
erf . . . 311
erfc . . . 312

exp . . . 313
exp2 . . . 314
expm1 . . . 315
fabs . . . 316
fdim . . . 317
floor . . . 318
fma . . . 319
fmax . . . 320
fmin . . . 321
fmod . . . 322
fpclassify . . . 323
frexp . . . 324
hypot . . . 325
ilogb . . . 326
isfinite . . . 327
isgreater . . . 328
isgreaterequal . . . 329
isinf . . . 330
isless . . . 331
islessequal . . . 332
islessgreater . . . 333
isnan . . . 334
isnormal . . . 335
isunordered . . . 336
ldexp . . . 337
lgamma . . . 338
log . . . 339
log10 . . . 340
log1p . . . 341
log2 . . . 342
logb . . . 343
lrint . . . 344
lround . . . 345
modf . . . 346
nan . . . 347
nearbyint . . . 348
nextafter . . . 349
nexttoward . . . 350
pow . . . 351
remainder . . . 353
remquo . . . 354
rint . . . 355

round . . . 356
scalbn and scalbln . . . 357
signbit . . . 359
sin . . . 360
sinh . . . 361
sqrt . . . 362
tan . . . 363
tanh . . . 364
tgamma . . . 365
trunc . . . 366

第 11 章 数値解析 — <numeric.h> 367

balance . . . 372
ccompanionmatrix . . . 375
cdeterminant . . . 376
cdiagonal . . . 378
cdiagonalmatrix . . . 381
cfevalarray . . . 383
cfunm . . . 385
charpolycoef . . . 387
choldecomp . . . 389
cinverse . . . 393
clinsolve . . . 395
cmean . . . 397
combination . . . 399
companionmatrix . . . 400
complexsolve . . . 401
complexsolvePP . . . 407
complexsolvePR . . . 409
complexsolveRP . . . 411
complexsolveRR . . . 413
complexsolveRRz . . . 416
condnum . . . 418
conv . . . 420
conv2 . . . 425
corrcoef . . . 429
correlation2 . . . 432
covariance . . . 434
cpolyeval . . . 437
cproduct . . . 439
cross . . . 441
csum . . . 442

ctrace . . . 444
ctriangularmatrix . . . 446
cumprod . . . 449
cumsum . . . 451
curvefit . . . 453
deconv . . . 457
derivative . . . 460
derivatives . . . 462
determinant . . . 465
diagonal . . . 467
diagonalmatrix . . . 470
difference . . . 472
dot . . . 473
eigen . . . 474
eigensystem . . . 479
expm . . . 480
factorial . . . 482
fevalarray . . . 483
fft . . . 485
filter . . . 491
filter2 . . . 496
findvalue . . . 498
fliplr . . . 500
flipud . . . 502
fminimum . . . 504
fminimums . . . 508
fsolve . . . 511
funm . . . 514
fzero . . . 516
gcd . . . 518
getnum . . . 520
hessdecomp . . . 521
histogram . . . 524
householdermatrix . . . 527
identitymatrix . . . 529
ifft . . . 530
integral1 . . . 532
integral2 . . . 535
integral3 . . . 537
integration2 . . . 539
integration3 . . . 541
interp1 . . . 544

interp2 . . . 546
inverse . . . 549
lcm . . . 551
lindata . . . 553
linsolve . . . 555
linspace . . . 557
llsqcovsolve . . . 560
llsqnonnegsolve . . . 562
llsqsolve . . . 564
logm . . . 567
logdata . . . 569
logspace . . . 571
ludecomp . . . 573
maxloc . . . 577
maxv . . . 579
mean . . . 580
median . . . 583
minloc . . . 585
minv . . . 586
norm . . . 587
nullspace . . . 591
oderk . . . 594
oderungekutta . . . 603
odesolve . . . 605
orthonormalbase . . . 607
pinverse . . . 610
polycoef . . . 615
polyder . . . 618
polyder2 . . . 620
polyeval . . . 623
polyevalarray . . . 625
polyevalm . . . 627
polyfit . . . 629
product . . . 633
qrdecomp . . . 635
qrdelete . . . 641
qrinsert . . . 645
rank . . . 649
rcondnum . . . 651
residue . . . 653
roots . . . 658
rot90 . . . 660

rsf2csf . . . 662
schurdecomp . . . 665
sign . . . 668
sort . . . 669
specialmatrix . . . 672
Cauchy . . . 672
ChebyshevVandermonde . . . 674
Chow . . . 675
Circul . . . 676
Clement . . . 677
DenavitHartenberg . . . 679
DenavitHartenberg2 . . . 681
Dramadah . . . 683
Fiedler . . . 685
Frank . . . 686
Gear . . . 687
Hadamard . . . 689
Hankel . . . 690
Hilbert . . . 691
InverseHilbert . . . 692
Magic . . . 693
Pascal . . . 694
Rosser . . . 696
Toeplitz . . . 697
Vandermonde . . . 698
Wilkinson . . . 699
sqrtm . . . 701
std . . . 703
sum . . . 705
svd . . . 707
trace . . . 714
triangularmatrix . . . 716
unwrap . . . 719
urand . . . 721
xcorr . . . 723

第 12 章 非ローカルジャンプ <setjmp.h> 726
setjmp . . . 728
longjmp . . . 730

第 13 章 シグナル処理 <signal.h> 731
signal . . . 733
raise . . . 736

第 14 章 可変長引数リスト — <stdarg.h> **737**
arraycopy . . . 741
va_arg . . . 743
va_arraydim . . . 744
va_arrayextent . . . 745
va_arraynum . . . 746
va_arraytype . . . 747
va_copy . . . 749
va_count . . . 751
va_datatype . . . 752
va_dim . . . 753
va_elementtype . . . 754
va_end . . . 755
va_extent . . . 756
va_start . . . 757
va_tagname . . . 760

第 15 章 ブール型とその値 <stdbool.h> **762**

第 16 章 共通定義 <stddef.h> **763**

第 17 章 入出力 <stdio.h> **765**
clearerr . . . 773
fclose . . . 774
feof . . . 775
ferror . . . 776
fflush . . . 777
fgetc . . . 778
fgetpos . . . 779
fgets . . . 781
fopen . . . 782
fprintf . . . 784
fputc . . . 791
fputs . . . 793
fread . . . 794
freopen . . . 795
fscanf . . . 796
fseek . . . 802
fsetpos . . . 804
ftell . . . 805
fwrite . . . 807
getchar . . . 809
getc . . . 810

gets . . . 812
perror . . . 813
printf . . . 815
putchar . . . 816
putc . . . 817
puts . . . 819
remove . . . 820
rename . . . 821
rewind . . . 823
scanf . . . 824
setbuf . . . 826
setvbuf . . . 827
snprintf . . . 829
sprintf . . . 830
sscanf . . . 831
tmpfile . . . 832
tmpnam . . . 833
ungetc . . . 835
vfprintf . . . 837
vprintf . . . 838
vsnprintf . . . 839
vsprintf . . . 840

第 18 章 汎用標準関数 — <stdlib.h> 841

abort . . . 844
abs . . . 845
labs . . . 845
atexit . . . 846
atoc . . . 848
atof . . . 850
atoi . . . 851
atol . . . 851
atoll . . . 851
bsearch . . . 853
calloc . . . 855
div . . . 856
exit . . . 857
free . . . 859
getenv . . . 860
iscnum . . . 861
isenv . . . 863
isnum . . . 864

iswnum . . . 865
malloc . . . 866
mblen . . . 868
mbstowcs . . . 870
mbtowc . . . 872
qsort . . . 874
rand . . . 876
realloc . . . 877
remenv . . . 879
srand . . . 880
strtod . . . 882
strtof . . . 882
strtold . . . 882
strtol . . . 885
strtoll . . . 885
strtoul . . . 885
strtoull . . . 885
system . . . 887
wcstombs . . . 888
wctomb . . . 890

第 19 章 文字列関数 — <string.h> 892

memchr . . . 894
memcmp . . . 895
memcpy . . . 896
memmove . . . 897
memset . . . 898
str2ascii . . . 899
str2mat . . . 900
stradd . . . 902
strcasecmp . . . 903
strcat . . . 904
strchr . . . 905
strcmp . . . 906
strcoll . . . 907
strconcat . . . 908
strcpy . . . 909
strcspn . . . 910
strdup . . . 911
strerror . . . 912
strgetc . . . 913
strjoin . . . 914

strlen . . . 915
strncasecmp . . . 916
strncat . . . 918
strncmp . . . 919
strncpy . . . 920
strpbrk . . . 921
strputc . . . 922
strrchr . . . 923
strrep . . . 924
strspn . . . 926
strstr . . . 927
strtok_r . . . 928
strtok . . . 930
strxfrm . . . 932

第 20 章 日付と時間の関数 — <time.h> 933

asctime . . . 936
clock . . . 938
ctime . . . 940
difftime . . . 941
gmtime . . . 942
localtime . . . 943
mktime . . . 944
strftime . . . 946
time . . . 950

第 21 章 拡張マルチバイトおよびワイド文字関数 — <wchar.h> 951

_wfopen . . . 955
btowc . . . 956
fgetwc . . . 958
fgetws . . . 960
fputwc . . . 962
fputws . . . 964
fwide . . . 966
getwc . . . 968
getwchar . . . 970
mbrlen . . . 971
mbrtowc . . . 973
mbsinit . . . 975
mbsrtowcs . . . 977
putwc . . . 979
putwchar . . . 981
ungetwc . . . 982

wcrtomb 984
wcscat 986
wcschr 988
wcscmp 990
wcscoll 992
wcscpy 994
wcscspn 995
wcsftime 997
wcslen 999
wcsncat 1000
wcsncmp 1002
wcsncpy 1004
wcspbrk 1006
wcsrchr 1008
wcsrtombs 1010
wcsspn 1012
wcsstr 1014
wcstod 1016
wcstok 1019
wcstol 1021
wcsxfrm 1023
wctob 1025
wmemchr 1027
wmemcmp 1029
wmemcpy 1031
wmemmove 1033
wmemset 1035

第 22 章 ワイド文字関数 — <wctype.h> **1037**

iswalnum 1040
iswalpha 1041
iswcntrl 1042
iswctype 1043
iswdigit 1045
iswgraph 1046
iswlower 1047
iswprint 1048
iswpunct 1049
iswspace 1050
iswupper 1051
iswxdigit 1052
towctrans 1053

towlower 1055
towupper 1056
wctrans 1057
wctype 1058

付 録 A 特定のプラットフォームでサポートされていない関数 1059
A.1 HPUX 1059
A.2 Linux 1062
A.3 Solaris 1066
A.4 Windows 1068

索引 1078

第1章　診断テスト — <assert.h>

ヘッダー **assert.h** には **assert** マクロが定義されていて、

NDEBUG

という別のマクロを参照しています。**NDEBUG** は **assert.h** では定義されて いません。ソースファイルで **assert.h** がインクルードされる箇所で **NDEBUG** がマクロとして定義されている場合、**assert** マクロは単純に次のように定義されます。

 # define assert (ignore) ((void)0)

assert マクロは、 **assert.h** がインクルードされるたびに、**NDEBUG** の現在の状態に従って再定義されます。

マクロ

asserth ヘッダーファイルには次のマクロが定義されています。

マクロ	説明
assert	プログラムに診断テストを挿入します。

移植性

このヘッダーには移植性に関する既知の問題はありません。

assert

構文

#include <**assert.h**>
void assert(**scalar** *expression*);

目的

プログラムに診断テストを挿入します。

戻り値

assert マクロは値を返しません。

パラメータ

expression 意図したとおりに動作しないとき false を返す条件式を指定します。

説明

assert マクロは診断テストをプログラムに挿入し、void 式に展開されます。**assert** マクロを実行すると、*expression* (スカラ型でなければならない) が false (つまり、比較結果が 0 に等しい) の場合は、**assert** マクロは失敗した特定の呼び出しに関する情報を標準出力に書き込みます (出力される情報には引数のテキスト、ソースファイルの名前、ソース行番号、失敗した動作を含む関数の名前が含まれ、あとの 3 つはプリプロセッサマクロ__**FILE**__および__**LINE**__と識別子 func それぞれの値です)。その後、 **abort** 関数を呼び出します。

例

```
/* a example for assert(). The program will
abort while i < 0 and display error message.*/
#include <assert.h>

int main() {
    int i;
    for(i = 5; ; i--)
    {
       printf("i= %d\n", i);
       assert(i >= 0);
    }
}
```

出力

```
i= 5
i= 4
i= 3
i= 2
i= 1
i= 0
i= -1
Assertion failed: i >= 0, file assert_func.c, line 10, function main()
```

関連項目
abort()

第2章　2Dおよび3Dプロット <chplot.h>

この章で説明されている高水準プロット機能は、Ch Professional Edition でのみ使用可能です。ヘッダーファイル **chplot.h** には、プロット **CPlot** クラス、このクラスに基づいた関数が含まれています。

プロットクラス **CPlot** により、Ch 言語環境内でのプロットの高水準な作成と操作が可能になります。**CPlot** は、アプリケーション、関数ファイル、および CGI プログラムを含む、多くのタイプの Ch プログラム内で直接使用することができます。プロットはデータ配列またはファイルから生成され、画面上に表示され、多くの異なるファイル形式で保存され、またはウェブ上で png または gif ファイル形式での stdout ストリームとして生成されます。

高水準プロット関数 **fplotxy**()、 **fplotxyz**()、**plotxy**()、 **plotxyf**()、 **plotxyz**()、 および **plotxyzf**() は容易に使用でき、プロットを迅速に作成できるように設計されています。これらの関数は、**CPlot** メンバ関数との結合で使用され、より複雑なプロットを作成します。

CPlot クラス

CPlot クラスは、Ch プログラムを通しての、2 次元（2D）および 3 次元（3D）プロットを作成に使用することができます。**CPlot** クラスのメンバ関数は、プロットエンジンを用いた実際のプロットを生成します。Gnuplot が、Ch 言語環境のこのリリースにおける表示用プロットエンジンとして、内部的に使用されます。

Public データ

なし。

Public メンバ関数

関数	説明
CPlot()	クラスコンストラクタ。クラスの新しいインスタンスを作成し、初期化します。
~CPlot()	クラスデストラクタ。クラスのインスタンスに関連付けられているメモリを解放します。
arrow()	矢印をプロットに追加します。
autoScale()	プロット軸の自動スケール調整を有効または無効にします。
axis()	2 次元プロットで、x-y 軸の描画を有効または無効にします。
axisRange()	プロット軸の範囲を設定します。

axes()	データセットの軸を指定します。
barSize()	エラーバーのサイズを設定します。
border()	プロット周囲の境界の描画を有効または無効にします。
borderOffsets()	プロット境界のプロットオフセットを設定します。
boundingBoxOrigin()	非推奨。代わりに CPlot::origin() を使用します。
boxBorder()	ボックス型のプロットで、境界の描画の有効と無効を切り替えます。
boxFill()	ボックスまたは曲線を単色またはパターンで塗りつぶします。
boxWidth()	ボックスの幅を設定します。
changeViewAngle()	3 次元プロットの視野角を変更します。
circle()	2 次元プロットに円を追加します。
colorBox()	3 次元サーフェスプロットのカラーボックスの描画を有効または無効にします。
contourLabel()	3 次元サーフェスプロットの輪郭ラベルを有効または無効にします。
contourLevels()	特定の位置に表示される 3 次元プロットに輪郭レベルを設定します。
contourMode()	3 次元サーフェスプロットに輪郭表示モードを設定します。
coordSystem()	3 次元プロットに座標系を設定します。
data()	2 次元、3 次元、または多次元のデータを **CPlot** クラスのインスタンスに追加します。
data2D()	1 つ以上の 2 次元のデータセットを **CPlot** クラスのインスタンスに追加します。
data2DCurve()	2 次元曲線のデータセットを **CPlot** クラスのインスタンスに追加します。
data3D()	1 つ以上の 3 次元のデータセットを **CPlot** クラスのインスタンスに追加します。
data3DCurve()	3 次元曲線のデータセットを **CPlot** クラスのインスタンスに追加します。
data3DSurface()	3 次元サーフェスのデータセットを **CPlot** クラスのインスタンスに追加します。
dataFile()	**CPlot** クラスのインスタンスにデータファイルを追加します。
dataSetNum()	**CPlot** クラスのインスタンス内の現在のデータセット番号を取得します。
deleteData()	以前に使用した **CPlot** クラスのインスタンスからデータを削除します。
deletePlots()	以前に使用した **CPlot** クラスのインスタンスからすべてのデータを削除し、オプションを既定値に再初期化します。
dimension()	2 次元または 3 次元にプロット次元を設定します。
displayTime()	プロットの現在の日時を表示します。
enhanceText()	特殊シンボルに対して拡張テキストを使用します。
func2D()	関数を使用して **CPlot** クラスのインスタンスに 2 次元のデータセットを追加します。
func3D()	関数を使用して **CPlot** クラスのインスタンスに 3 次元のデータセットを追加します。
funcp2D()	パラメータを指定した関数を使用して **CPlot** クラスのインスタンスに 2 次元のデータセットを追加します。
funcp3D()	パラメータを指定した関数を使用して **CPlot** クラスのインスタンスに 3 次元のデータセットを追加します。
getLabel()	軸のラベルを取得します。
getOutputType()	プロット出力の種類を取得します。
getSubplot()	サブプロットの要素へのポインタを取得します。
getTitle()	プロットのタイトルを取得します。

grid() グリッドの表示を有効または無効にします。
isUsed() **CPlot** クラスのインスタンスが使用されたかどうかをテストします。
label() 軸ラベルを設定します。
legend() データセットの凡例を追加します。
legendLocation() プロットの凡例の位置を指定します。
legendOption() プロットの凡例のオプションを設定します。
line() プロットに線を追加します。
lineType() 線の種類、幅、色、縦線、ステップなどを設定します。
margins() プロットにマージンを設定します。
origin() プロットの境界ボックスの原点の位置を設定します。
outputType() プロット出力の種類を設定します。
plotType() プロットの種類を設定します。
plotting() **CPlot** クラスのインスタンスからプロットを生成します。
point() プロットに線を追加します。
pointType() 点の種類、サイズおよび色を設定します。
polarPlot() 2 次元プロットに極座標系を使用するように設定します。
polygon() プロットに多角形を追加します。
rectangle() 2 次元プロットに長方形を追加します。
removeHiddenLine() 3 次元プロットの陰線消去を有効または無効にします。
scaleType() プロットの軸のスケールの種類を設定します。
showMesh() 3 次元プロットのメッシュの表示を有効または無効にします。
size() プロット自体を出力ファイルまたはキャンバスのサイズ応じて決めます。
size3D() 3 次元プロットのサイズを変更します。
sizeOutput() 出力ファイルのサイズを変更します。
sizeRatio() プロットのアスペクト比を変更します。
smooth() データの補間と近似でプロット曲線を滑らかにします。
subplot() サブプロットのグループを作成します。
text() プロットにテキスト文字列を追加します。
tics() 軸のチックマークの表示を有効または無効にします。
ticsDay() 軸のチックマークのラベルを曜日単位に設定します。
ticsDirection() 軸のどの方向にチックマークが描かれるかを設定します。
ticsFormat() チックラベルの数の書式を設定します。
ticsLabel() 任意の軸ラベルについて、表示位置とテキストラベルを設定します。
ticsLevel() 3 次元プロットで、チックマーク描画時の z-軸オフセットを設定します。
ticsLocation() 軸のチックマークの位置が、境界または軸の上になるように指定します。
ticsMirror() 反対側の軸における軸のチックマークの表示を有効または無効にします。
ticsMonth() 軸のチックマークのラベルを月単位に設定します。
ticsPosition() 軸の指定された位置にチックマークを追加します。
ticsRange() 軸に対するチックの範囲を指定します。
title() プロットタイトルを設定します。

マクロ

以下のマクロが **CPlot** クラスに対して定義されています。

マクロ	説明
PLOT_ANGLE_DEG	角度の値の単位を度（degree）に選択します。
PLOT_ANGLE_RAD	角度の値の単位をラジアンに選択します。
PLOT_AXIS_X	x 軸のみを選択します。
PLOT_AXIS_X2	上端の x2 軸のみを選択します。
PLOT_AXIS_XY	x および y 軸を選択します。
PLOT_AXIS_XYZ	x、y、および z 軸を選択します。
PLOT_AXIS_Y	y 軸のみを選択します。
PLOT_AXIS_Y2	右端の y2 軸のみを選択します。
PLOT_AXIS_Z	z 軸のみを選択します。
PLOT_BORDER_BOTTOM	プロットの下端。
PLOT_BORDER_LEFT	プロットの左端。
PLOT_BORDER_TOP	プロットの上端。
PLOT_BORDER_RIGHT	プロットの右端。
PLOT_BORDER_ALL	プロットの上下左右すべての境界。
PLOT_BOXFILL_EMPTY	ボックスを塗りつぶしません。
PLOT_BOXFILL_SOLID	ボックスを単色で塗りつぶします。
PLOT_BOXFILL_PATTERN	ボックスをパターンで塗りつぶします。
PLOT_CONTOUR_BASE	サーフェスプロット用の輪郭線を xy 平面上に描画します。
PLOT_CONTOUR_SURFACE	サーフェスプロット用の輪郭線を表面上に描画します。
PLOT_COORD_CARTESIAN	3 次元プロットに対してデカルト座標系を使用します。
PLOT_COORD_CYLINDRICAL	3 次元プロットに対して円筒座標系を使用します。
PLOT_COORD_SPHERICAL	3 次元プロットに対して球面座標系を使用します。
PLOT_OFF	オプションを無効にするフラグ。
PLOT_ON	オプションを有効にするフラグ。
PLOT_OUTPUTTYPE_DISPLAY	プロットを画面上に表示します。
PLOT_OUTPUTTYPE_FILE	プロットをファイルに出力します。
PLOT_OUTPUTTYPE_STREAM	プロットを stdout ストリームとして出力します。
PLOT_PLOTTYPE_BOXERRORBARS	**PLOT_PLOTTYPE_BOXES** と **PLOT_PLOTTYPE_YERRORBARS** のプロットの種類の組み合わせ。
PLOT_PLOTTYPE_BOXES	指定した x 座標を中心としたボックスを描画します。
PLOT_PLOTTYPE_BOXXYERRORBARS	**PLOT_PLOTTYPE_BOXES** と **PLOT_PLOTTYPE_XYERRORBARS** のプロットの種類の組み合わせ。
PLOT_PLOTTYPE_CANDLESTICKS	財務または統計データの箱ひげ図のプロットを表示します。
PLOT_PLOTTYPE_DOTS	各データ点のマーキングにドットを使用します。

PLOT_PLOTTYPE_FILLEDCURVES	曲線で囲まれた領域を単色またはパターンで塗りつぶします。
PLOT_PLOTTYPE_FINANCEBARS	財務データを表示します。
PLOT_PLOTTYPE_FSTEPS	近接する点が 2 つの線分、1 つは (x1,y1) から (x1,y2)、もう 1 つは (x1,y2) から (x2,y2) で結ばれます。
PLOT_PLOTTYPE_HISTEPS	点 x1 は ((x0+x1)/2,y1) から ((x1+x2)/2,y1) の水平線で表されます。 近接する線は ((x1+x2)/2,y1) から ((x1+x2)/2,y2) の縦線で結ばれます。
PLOT_PLOTTYPE_IMPULSES	（2 次元プロットに対して）x 軸から、または（3 次元プロットに対して）xy 平面からデータ点への縦線を表示します。
PLOT_PLOTTYPE_LINES	データ点は、線で接続されます。
PLOT_PLOTTYPE_LINESPOINTS	各データ点にマーカーが表示され、線で結ばれます。
PLOT_PLOTTYPE_POINTS	各データ点にマーカーが表示されます。
PLOT_PLOTTYPE_STEPS	近接する点は 2 つの線分、1 つは (x1,y1) から (x2,y1)、もう 1 つは (x2,y1) から (x2,y2) で結ばれます。
PLOT_PLOTTYPE_SURFACES	データ点が結ばれて、表面のようになめらかになります。3 次元プロットのみ。
PLOT_PLOTTYPE_VECTORS	ベクトルを表示します。
PLOT_PLOTTYPE_XERRORBARS	水平の誤差表示線がある点線を表示します。
PLOT_PLOTTYPE_XERRORLINES	水平の誤差線がある折れ線と点を表示します。
PLOT_PLOTTYPE_XYERRORBARS	水平および垂直の誤差表示線がある点線を表示します。
PLOT_PLOTTYPE_XYERRORLINES	水平および垂直の誤差線がある折れ線と点を表示します。
PLOT_PLOTTYPE_YERRORBARS	垂直の誤差表示線がある点線を表示します。
PLOT_PLOTTYPE_YERRORLINES	垂直の誤差線がある折れ線と点を表示します。
PLOT_SCALETYPE_LINEAR	指定した軸に対して線形目盛を使用します。
PLOT_SCALETYPE_LOG	指定した軸に対して対数目盛を使用します。
PLOT_TEXT_CENTER	指定した点にテキストの中心を置きます。
PLOT_TEXT_LEFT	指定した点にテキストの左側を合わせます。
PLOT_TEXT_RIGHT	指定した点にテキストの右側を合わせます。
PLOT_TICS_IN	チックマークを内側に描画します。
PLOT_TICS_OUT	チックマークを外側に表示します。

関数

以下の関数が、**CPlot** クラスを用いて実装されています。

関数	説明
fplotxy()	指定した範囲の *x* の 2 次元関数をプロットするか、または **CPlot** クラスの

	インスタンスを初期化します。
fplotxyz()	指定した範囲の *x* と *y* の 3 次元関数をプロットするか、または **CPlot** クラスのインスタンスを初期化します。
plotxy()	2 次元データセットをプロットするか、または **CPlot** クラスのインスタンスを初期化します。
plotxyf()	ファイルから取得した 2 次元データをプロットするか、または **CPlot** クラスのインスタンスを初期化します。
plotxyz()	3 次元データセットをプロットするか、または **CPlot** クラスのインスタンスを初期化します。
plotxyzf()	ファイルから取得した 3 次元データをプロットするか、または **CPlot** クラスのインスタンスを初期化します。

リファレンス

T. Williams, C. Kelley, D. Denholm, D. Crawford, et al., *Gnuplot — An Interactive plotting Program*, Version 3.7, December 3, 1998, ftp://ftp.gnuplot.vt.edu/.

Gnuplot の著作権情報

```
 * software.
 * Permission to distribute the released version of the source code
 * along with corresponding source modifications in the form of a patch
 * file is granted with same provisions 2 through 4 for binary
 * distributions.
 *
 * This software is provided "as is" without express or implied warranty
 * to the extent permitted by applicable law.
]*/
```

CPlot::arrow

Synopsis
#include <**chplot.h**>
void arrow(**double** *x_head*, **double** *y_head*, **double** *z_head*, **double** *x_tail*, **double** *y_tail*, **double** *z_tail*, ...
/* [**char** *option*] */);

Syntax
arrow(*x_head*, *y_head*, *z_head*, *x_tail*, *y_tail*, *z_tail*)
arrow(*x_head*, *y_head*, *z_head*, *x_tail*, *y_tail*, *z_tail*, *option*)

Purpose
Add an arrow to a plot.

Return Value
None.

Parameters
x_head The x coordinate of the head of the arrow.

y_head The y coordinate of the head of the arrow.

z_head For 2D plots this is ignored. For 3D plots, the z coordinate of the head of the arrow.

x_tail For x coordinate of the tail of the arrow.

y_tail The y coordinate of the tail of the arrow.

z_tail For 2D plots this is ignored. For 3D plots, the z coordinate of the tail of the arrow.

option The option for the arrow.

Description
This function adds an arrow to a plot. The arrow points from (*x_tail*, *y_tail*, *z_tail*) to (*x_head*, *y_head*, *z_head*). These coordinates are specified using the same coordinate system as the curves of the plot.

An arrow is not counted as a curve. Therefore, it does not affect the number of legends added by **CPlot**::**legend**(legend, num).

The optional argument `option` of string type with the following values can be used to fine tune the arrow based on the argument for set arrow command of the gnuplot.

```
{ {nohead | head | backhead | heads}
  {size <length>,<angle>{,<backangle>}}
  {filled | empty | nofilled}
  {front | back}
```

```
{ {linestyle | ls <line_style>}
  | {linetype | lt <line_type>}
    {linewidth | lw <line_width} } } }
```

Specifying ‘nohead‘ produces an arrow drawn without a head—a line segment. This gives you yet another way to draw a line segment on the plot. By default, an arrow has a head at its end. Specifying ‘backhead‘ draws an arrow head at the start point of the arrow while ‘heads‘ draws arrow heads on both ends of the line. Not all terminal types support double-ended arrows.

Head size can be controlled by `'size <length>,<angle>'` or `'size <length>,<angle>,<backangle>'`, where `'<length>'` defines length of each branch of the arrow head and ‘¡angle¿‘ the angle (in degrees) they make with the arrow. `'<Length>'` is in x-axis units; this can be changed by ‘first‘, ‘second‘, ‘graph‘, ‘screen‘, or ‘character‘ before the ¡length¿; see ‘coordinates‘ for details. ‘¡Backangle¿‘ only takes effect when ‘filled‘ or ‘empty‘ is also used. Then, `'<backangle>'` is the angle (in degrees) the back branches make with the arrow (in the same direction as `'<angle>'`). The ‘fig‘ terminal has a restricted backangle function. It supports three different angles. There are two thresholds: Below 70 degrees, the arrow head gets an indented back angle. Above 110 degrees, the arrow head has an acute back angle. Between these thresholds, the back line is straight.

Specifying ‘filled‘ produces filled arrow heads (if heads are used). Filling is supported on filled-polygon capable terminals, otherwise the arrow heads are closed but not filled. The same result (closed but not filled arrow head) is reached by specifying ‘empty‘.

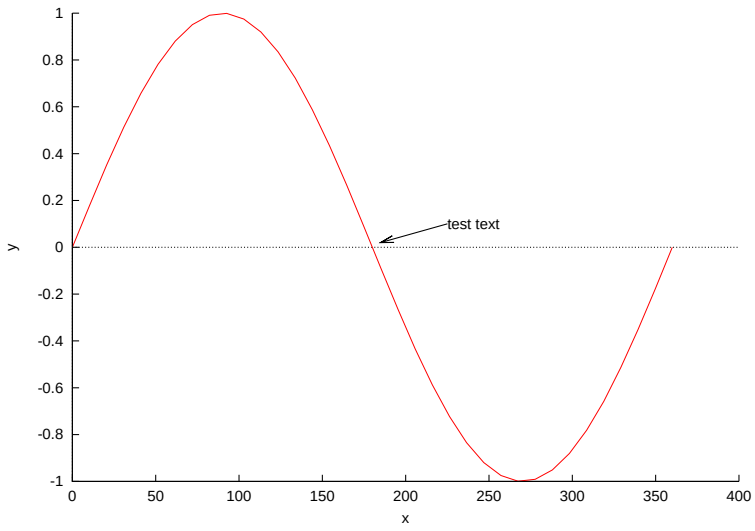
If ‘front‘ is given, the arrow is written on top of the graphed data. If ‘back‘ is given (the default), the arrow is written underneath the graphed data. Using ‘front‘ will prevent an arrow from being obscured by dense data.

The ‘linetype‘ is followed by an integer index representing the line type for drawing. The line type varies depending on the terminal type used (see **CPlot**::**outputType**). Typically, changing the line type will change the color of the line or make it dashed or dotted. All terminals support at least six different line types. The ‘linewidth‘ is followed by a scaling factor for the line width. The line width is ‘linewidth‘ multiplied by the default width. Typically the default width is one pixel.

Example 1
Compare with output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: arrow_1.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.arrow(185, 0.02, 0, 225, 0.1, 0);
    plot.text("test text", PLOT_TEXT_LEFT, 225, 0.1, 0);
    plot.data2D(x, y);
    plot.plotting();
}
```

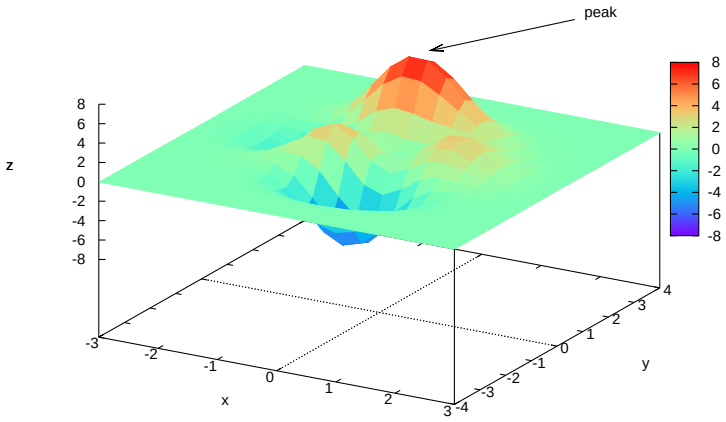
Output



Example 2
Compare with the output for examples in **CPlot**::**data3D**() and **CPlot**::**data3DSurface**().

```
/* File: arrow_2.ch */
#include <math.h>
#include <chplot.h>

int main() {
    double x[20], y[30], z[600];
    int i,j;
    class CPlot plot;

    lindata(-3, 3, x);
    lindata(-4, 4, y);
    for(i=0; i<20; i++) {
        for(j=0; j<30; j++) {
            z[30*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.arrow(0, 2, 8, 2, 3, 12);
    plot.text("peak", PLOT_TEXT_LEFT, 2.1, 3.15, 12.6);
    plot.plotting();

}
```

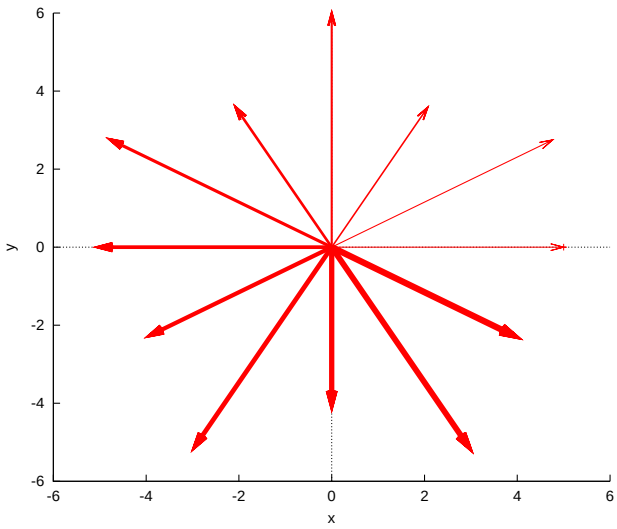
Output



Example 3

```
/* File: arrow_3.ch */
#include <math.h>
#include <chplot.h>

int main() {
    double complex z[12];
    char option[64];
    int i;
    class CPlot plot;

    for (i=0; i<12; i++) {
        real(z[i]) = (5+sin(150*i*M_PI/180))*cos(30*i*M_PI/180);
        imag(z[i]) = (5+sin(150*i*M_PI/180))*sin(30*i*M_PI/180);
        sprintf(option, "linetype 1 linewidth %d", i+1);
        plot.arrow(real(z[i]), imag(z[i]), 0, 0, 0, 0, option);
    }
    plot.axisRange(PLOT_AXIS_XY, -6, 6);   /* one point cannot do autorange */
    plot.point(real(z[0]), imag(z[0]), 0); /* CPlot::arrow() itself is not a data set */
    plot.plotType(PLOT_PLOTTYPE_POINTS, 0);
    plot.sizeRatio(-1);
    plot.plotting();
}
```

Output



Example 4
See an example on page 149 for **CPlot**::**plotType**() on how `option` is used in comparison with the plot type **PLOT_PLOTTYPE_VECTORS**.

See Also
CPlot::**circle**(), **CPlot**::**data2D**(), **CPlot**::**outputType**(), **CPlot**::**plotType**(), **CPlot**::**point**(), **CPlot**::**polygon**(), **CPlot**::**rectangle**().

CPlot::autoScale

Synopsis
#include <**chplot.h**>
void autoScale(**int** *axis*, **int** *flag*);

Purpose
Set autoscaling of axes on or off.

Return Value
None.

Parameters
axis The axis to be set. Valid values are:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

flag Enable or disable auto scaling.

PLOT_ON The option is enabled.

PLOT_OFF The option is disabled.

Description

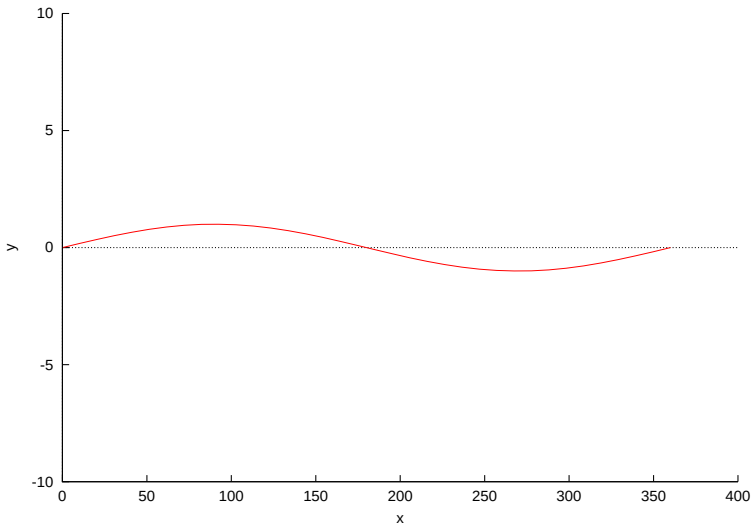
Autoscaling of the axes can be **PLOT_ON** or **PLOT_OFF**. Default is **PLOT_ON**. If autoscaling for an axis is disabled, an axis range of [-10:10] is used.

Example

Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: autoScale.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.autoScale(PLOT_AXIS_Y, PLOT_OFF);
    plot.data2D(x, y);
    plot.plotting();
}
```

Output



CPlot::axis

Synopsis
#include <**chplot.h**>
void axis(int *axis*, **int** *flag*);

Purpose
Enable or disable drawing of x-y axis on 2D plots.

Return Value
None.

Parameters
axis The *axis* parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_XY Select the x and y axes.

flag This parameter can be set to:

PLOT_ON Enable drawing of the specified axis.

PLOT_OFF Disable drawing of the specified axis.

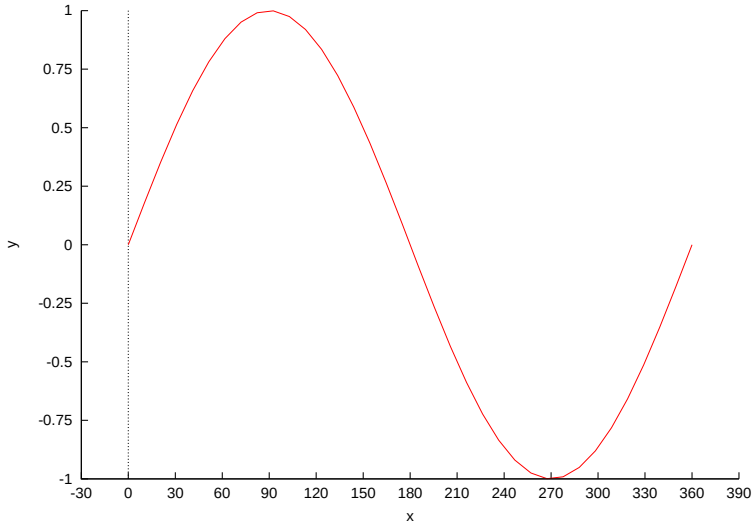
Description
Enable or disable the drawing of the x-y axes on 2D plots. By default, the x and y axes are drawn.

Example
Compare with the output for the example in **CPlot**::**axisRange**().

```
/* File: axis.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    int i;
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);                        // Y-axis data.
    plot.axisRange(PLOT_AXIS_X, -30, 390);
    plot.ticsRange(PLOT_AXIS_X, 30, -30, 390);
    plot.axisRange(PLOT_AXIS_Y, -1, 1);
    plot.ticsRange(PLOT_AXIS_Y, .25, -1, 1);
```

```
    plot.axis(PLOT_AXIS_X, PLOT_OFF);
    plot.data2D(x, y);
    plot.plotting();
}
```

Output



CPlot::axisRange

Synopsis
#include <**chplot.h**>
void axisRange(**int** *axis*, **double** *minimum*, **double** *maximum*, ... /* [**double** *incr*] */);

Syntax
axisRange(*axis*, *minimum*, *maximum*)
axisRange(*axis*, *minimum*, *maximum*, *incr*)

Purpose
Specify the range for an axis.

Return Value
None.

Parameters
axis The *axis* parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

minimum The axis minimum.

maximum The axis maximum.

incr The increment between tic marks. By default or when *incr* is 0, the increment between tic marks is calculated internally.

Description

The range for an axis can be explicitly specified with this function. Autoscaling for the specified axis is disabled and any previously specified labeled tic-marks are overridden. If the axis is logarithmic specified by the member function **scaleType**(), the increment will be used as a multiplicative factor.

Note that

```
plot.axisRange(axis, min, max, incr);
```

is obsolete. Use

```
plot.axisRange(axis, min, max);
plot.ticsRange(axis, incr);
```

or

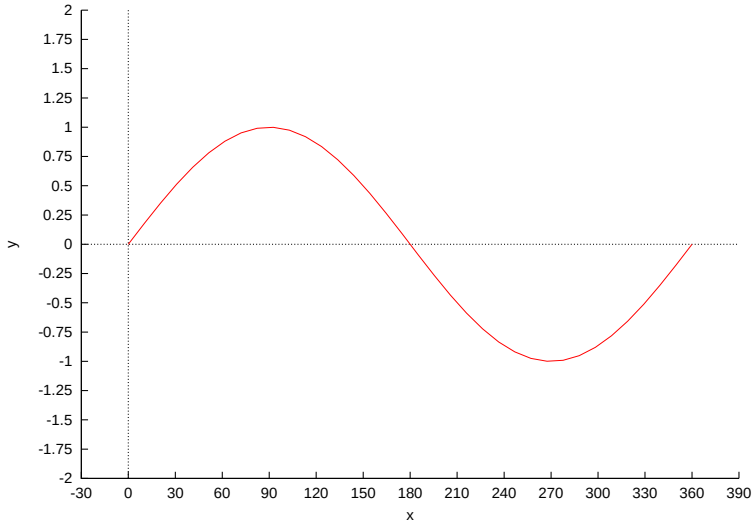
```
plot.ticsRange(axis, incr, start);
plot.ticsRange(axis, incr, start, end);
```

Example 1

Compare with the output for the examples in **CPlot**::**axis**() and **CPlot**::**grid**().

```
/* File: axisRange.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);                        // Y-axis data.
    plot.axisRange(PLOT_AXIS_X, -30, 390);
    plot.ticsRange(PLOT_AXIS_X, 30);
    plot.axisRange(PLOT_AXIS_Y, -2, 2);
    plot.ticsRange(PLOT_AXIS_Y, 0.25);
    plot.data2D(x, y);
    plot.plotting();
}
```

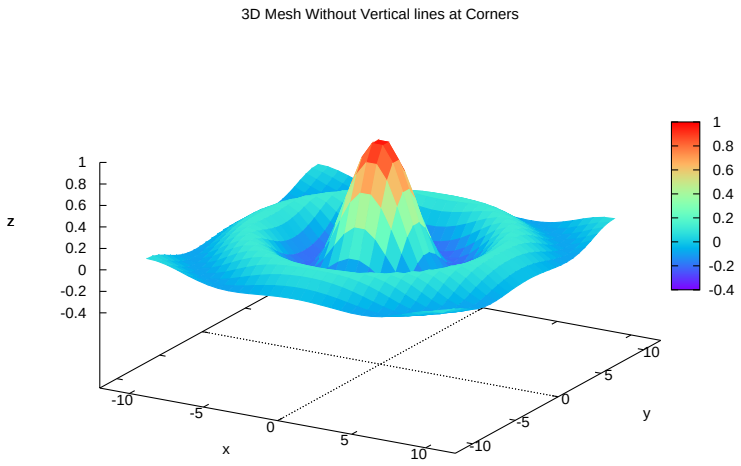
Output



Example 2
3D mesh plot without vertical lines at the corners. Compare with the output for examples in **CPlot**::**data3D**() and **CPlot**::**data3DSurface**().

```
/* File: axisRange_2.ch */
#include <chplot.h>
#include <math.h>

int main() {
    double x[30], y[30], z[900];
    double r;
    int i, j;
    class CPlot plot;

    lindata(-10, 10, x);
    lindata(-10, 10, y);
    for(i=0; i<30; i++) {
        for(j=0; j<30; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[30*i+j] = sin(r)/r;
        }
    }
    plot.data3D(x, y, z);
    plot.axisRange(PLOT_AXIS_XY, -12, 12);
    plot.title("3D Mesh Without Vertical lines at Corners");
    plot.plotting();
}
```

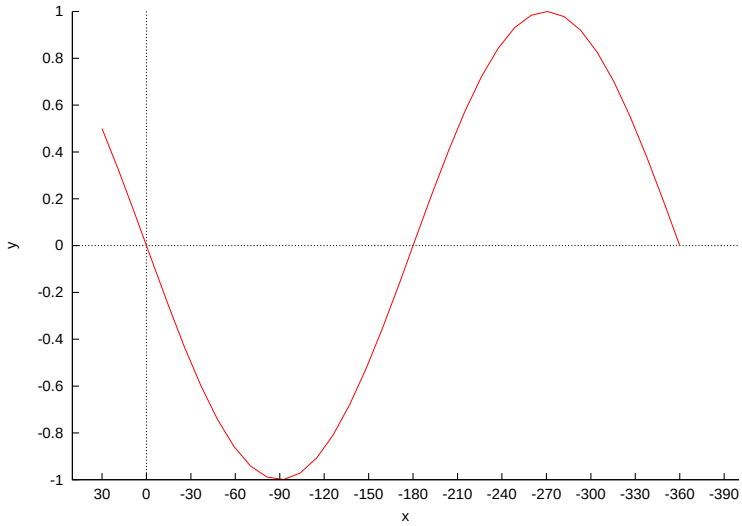
Output



Example 3
X axis range is reversed.

```
/* File: axisRange_3.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(-360, 30, x);
    y = sin(x*M_PI/180);                          // Y-axis data.
    plot.axisRange(PLOT_AXIS_X, 50, -400);
    plot.ticsRange(PLOT_AXIS_X, -30);
    plot.data2D(x, y);
    plot.plotting();
}
```

Output



See Also
CPlot::**ticsRange**(), **CPlot**::**autoScale**(), **CPlot**::**ticsLabel**().

CPlot::axes

Synopsis
#include <**chplot.h**>
void axes(**int** *num*, **string_t** *axes*);

Syntax
axes(*num*, *axes*)

Purpose
Specify the axes for a data set.

Return Value
None.

Parameters
num The data set the axes are specified.

axes The axes for the specified data set.

Description

A **CPlot** class lets you use each of the four borders – x (bottom), x2 (top), y (left) and y2 (right) – as an independent axis. The **axes**() function lets you choose which pair of axes a given set of data specified in `num` is plotted against.

There are four possible sets of axes available. The argument `axes` is used to select the axes for which a particular line should be scaled. The string `"x1y1"` refers to the axes on the bottom and left; `"x2y2"` to those on the top and right; `"x1y2"` to those on the bottom and right; and `"x2y1"` to those on the top and left.

Other options such as labels and ranges can be specified other member functions by selecting a proper axis with one of following macros.

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

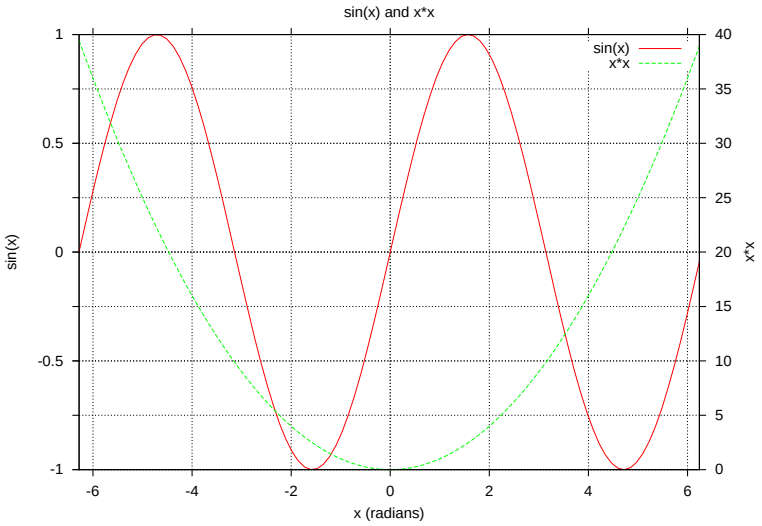
PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

Example 1

```
/* File: axes.ch */
#include <chplot.h>
#include <math.h>

double func1(double x) {
   return sin(x);
}
double func2(double x) {
   return x*x;
}

int main() {
    class CPlot plot;
    double x0= -6.28, xf = 6.24;
    int num = 100;

    plot.title("sin(x) and x*x");
    plot.label(PLOT_AXIS_X, "x (radians)");
    plot.label(PLOT_AXIS_Y, "sin(x)");
    plot.label(PLOT_AXIS_Y2, "x*x");
    plot.axisRange(PLOT_AXIS_X, x0, xf);
    plot.ticsMirror(PLOT_AXIS_X, PLOT_ON);
    plot.axisRange(PLOT_AXIS_X2, x0, xf);
    plot.ticsRange(PLOT_AXIS_Y, 0.5);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_Y2, PLOT_ON);
    plot.ticsDirection(PLOT_TICS_OUT);
    plot.tics(PLOT_AXIS_Y2, PLOT_ON);
    plot.ticsRange(PLOT_AXIS_Y2, 5);
    plot.grid(PLOT_ON, "x y y2");
    plot.func2D(x0, xf, num, func1);
    plot.legend("sin(x)", 0);
    plot.func2D(x0, xf, num, func2);
    plot.legend("x*x", 1);
    plot.axes(1, "x2y2");
    plot.plotting();
}
```

Output



Example 2
See an example on page 153 for plot type **PLOT_PLOTTYPE_FINANCEBARS** in **CPlot**:**plotType**(). In that example, the y2 axis is used to display different data.

See Also
CPlot::**legend**(), **CPlot**::**axisRange**().

CPlot::barSize

Synopsis
#include <**chplot.h**>
void barSize(**double** *size*);

Syntax
barSize(*size*)

Purpose
Set the size of error bars.

Return Value
None.

Parameters
size The size of error bars. The value for `size` is in the range [0, 1.0].

Description
This function specifies the size of of error bars for a plot type of **PLOT_PLOTTYPE_XERRORBARS**, **PLOT_PLOTTYPE_XYERRORBARS**, and

PLOT_PLOTTYPE_YERRORBARS. specified by member function **CPlot**::**plotType**(). The value for `size` is in the range [0, 1.0]. The value 0 is for no error bar. The value 1.0 is for the largest error bar.

Example

See an example on page 153 for the plot type **PLOT_PLOTTYPE_XYERRORBARS** in **CPlot**::**plotType**().

See Also

CPlot::**boxWidth**(), **CPlot**::**plotType**().

CPlot::border()

Synopsis

#include <**chplot.h**>

void border(**int** *location*, **int** *flag*);

Purpose

Enable or disable display of a bounding box around the plot.

Return Value

None.

Parameter

location The location of the effected border segment.

PLOT_BORDER_BOTTOM The bottom of the plot.

PLOT_BORDER_LEFT The left side of the plot.

PLOT_BORDER_TOP The top of the plot.

PLOT_BORDER_RIGHT The right side of the plot.

PLOT_BORDER_ALL All sides of the plot.

flag Enable or disable display of a box around the boundary or the plot.

PLOT_ON Enable drawing of the box.

PLOT_OFF Disable drawing of the box.

Description

This function turns the display of a border around the plot on or off. By default, the border is drawn on the left and bottom sides for 2D plots, and on all sides for 3D plots.

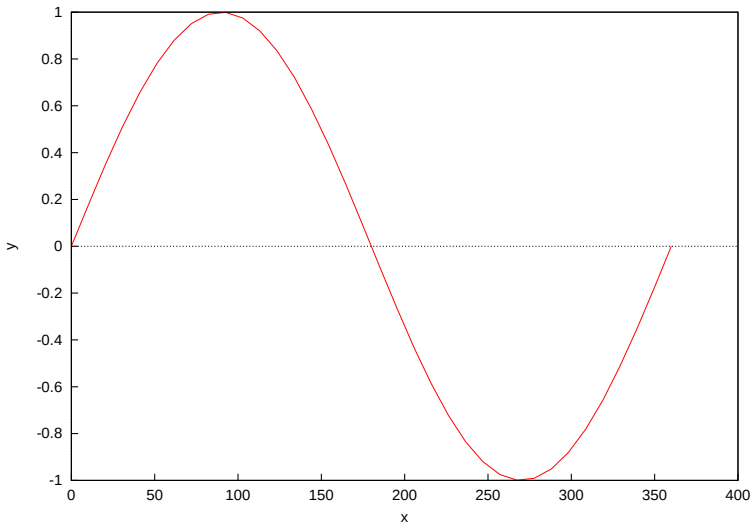
Example 1

Compare with the example output in **CPlot**::**ticsMirror**().

```
/* File: border.ch */
#include <math.h>
#include <chplot.h>
```

```
int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, tetha);
    y = sin(x*M_PI/180);
    plot.data2D(x, y);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.plotting();
}
```

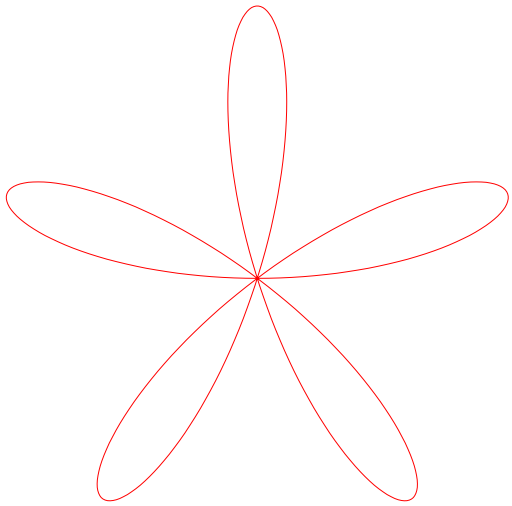
Output



Example 2
Compare with the example output in **CPlot**::**polarPlot**().

```
/* File: border_2.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 360;
    array double theta[numpoints], r[numpoints];
    class CPlot plot;

    lindata(0, M_PI, theta);
    r = sin(5*theta);                   // Y-axis data.
    plot.polarPlot(PLOT_ANGLE_RAD);
    plot.data2D(theta, r);
    plot.label(PLOT_AXIS_XY, NULL);
    plot.sizeRatio(1);
    plot.border(PLOT_BORDER_ALL, PLOT_OFF);
    plot.tics(PLOT_AXIS_XY, PLOT_OFF);
    plot.axis(PLOT_AXIS_XY, PLOT_OFF);
    plot.plotting();
}
```

Output



CPlot::borderOffsets

Synopsis
#include <**chplot.h**>
void borderOffsets(**double** *left*, **double** *right*, **double** *top*, **double** *bottom*);

Purpose
Specify the size of offsets around the data points of a 2D plot in the same units as the plot axis.

Return Value
None.

Parameters
left The offset on the left side of the plot.

right The offset on the right side of the plot.

top The offset on the top of the plot.

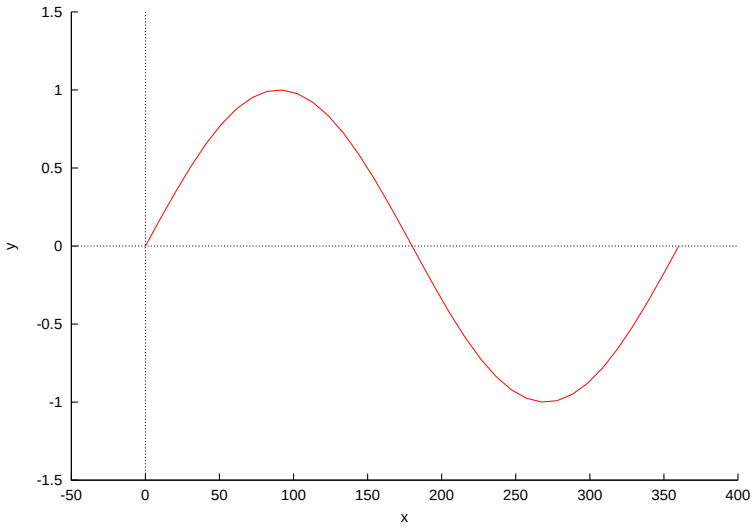
bottom The offset on the bottom of the plot.

Description
For 2D plots, this function specifies the size of offsets around the data points of an autoscaled plot. This function can be used to create empty space around the data. The *left* and *right* arguments are specified in the units of the x-axis. The *top* and *bottom* arguments are specified in the units of the y-axis.

Example
Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: borderOffsets.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.data2D(x, y);
    plot.borderOffsets(30, 30, .5, .25);
    plot.plotting();
}
```

Output



See Also
CPlot::**autoScale**(), **CPlot**::**axisRange**(), **CPlot**::**ticsLabel**(), **CPlot**::**margins**().

CPlot::boxBorder

Synopsis
#include <**chplot.h**>
void boxBorder(**int** *num*, ... /* [**int** *linetype*] */);

Syntax
boxBorder(*num*)
boxBorder(*num*, *linetype*)

Purpose
Enable or disable drawing of a border for a plot of box type.

Return Value
None.

Parameters
num The data set of plot type of box or filled curve to be bounded with a border. If `num` is -1, the border handling will be applied to all boxes and filled curves for the plot.

linetype The *linetype* parameter specifies the line type used to draw bounding lines.

Description
This function specifies how borders for boxes and filled curves will handled for a plot type of **PLOT_PLOTTYPE_BOXES**, **PLOT_PLOTTYPE_BOXERRORBARS**, **PLOT_PLOTTYPE_BOXXYERRORBARS**, **PLOT_PLOTTYPE_CANDLESTICKS**, and **PLOT_PLOTTYPE_FILLEDCURVES** specified by member function **CPlot**::**plotType**(). By default, the box or filled curve is bounded by a solid line of the current line type. If the line type is not specified by the function call of **boxBorder**(*num*), no bounding lines are drawn.

Example
See an example on page 30 for **CPlot**:**boxFill**().

See Also
CPlot::**boxFill**(), **CPlot**::**boxWidth**(), **CPlot**::**plotType**().

CPlot::boxFill

Synopsis
#include <**chplot.h**>
void boxFill(**int** *num*, **int** *style*, ... /* [**double** *density*], **int** *patternnum*] */);

Syntax
boxFill(*num*, *PLOT_BOXFILL_EMPTY*)
boxFill(*num*, *PLOT_BOXFILL_SOLID*)
boxFill(*num*, *PLOT_BOXFILL_SOLID*, `density`)
boxFill(*num*, *PLOT_BOXFILL_PATTERN*)
boxFill(*num*, *PLOT_BOXFILL_PATTERN*, *patternnum*)

Purpose
Fill a box or filled curve with a solid color or pattern.

Return Value
None.

Parameters

num The data set of plot type of box or filled curve to be filled. If `num` is -1, the fill style will be applied to all boxes and filled curves for the plot.

style The *style* parameter can take one of the following values:

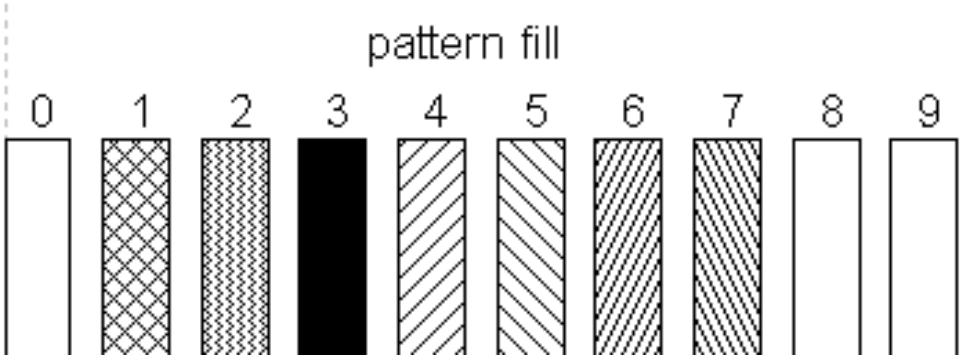
PLOT_BOXFILL_EMPTY Do not fill the box. The default is to have an empty box.

PLOT_BOXFILL_SOLID Fill the boxes or filled curves with a solid color.

PLOT_BOXFILL_PATTERN Fill the boxes or filled curves with a pattern.

density The density of solid color in the range of [0, 1.0]. If the value for `density` is 0.0, the box is empty. If it is 1.0, the inner area is of the same color as the current line type. If no `density` parameter is given, it defaults to 1.0.

patternnum The parameter`patternnum` option causes filling to be done with a fill pattern supplied by the terminal driver. The kind and number of available fill patterns depend on the terminal driver. If multiple datasets using filled boxes are plotted, the pattern cycles through all available pattern types, starting from `patternnum`, much as the line type cycles for multiple line plots. The available patterns in Windows are shown below.



Description

This function specifies how boxes and filled curves are filled with a solid color or pattern. The fill style can be applied to plot types of **PLOT_PLOTTYPE_BOXES**, **PLOT_PLOTTYPE_BOXERRORBARS**, **PLOT_PLOTTYPE_BOXXYERRORBARS**, **PLOT_PLOTTYPE_CANDLESTICKS**, and **PLOT_PLOTTYPE_FILLEDCURVES** specified by member function **CPlot**::**plotType**().

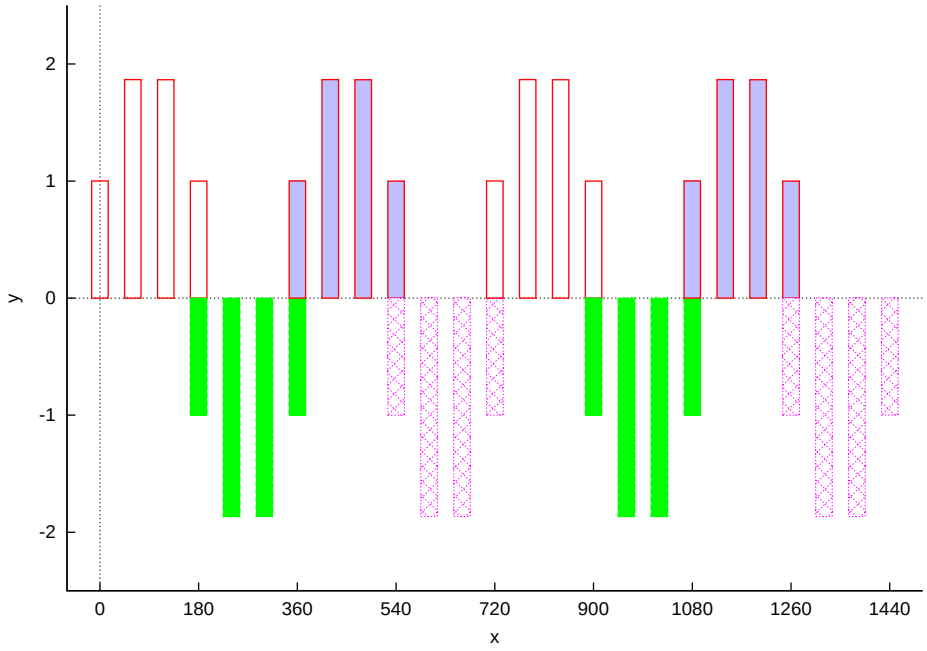
Example

In this example, each box has width of 30 specified by **CPlot**::**boxWidth**(). The first box is by default empty. The second box is filled with a solid color. The third one is filled with a solid color of density 0.25. The border of this box uses the line type for the first box specified by **CPlot**::**boxBorder**(). The fourth box is filled with a pattern. The border of this box is empty. The four boxes are repeated twice by a outer loop with index j.

```
/* File: boxFill.ch */
#include <math.h>
#include <chplot.h>
#include <numeric.h>
```

```
#define N 4
#define M 2
int main() {
    int i, j, k, numpoints = 4, linetype, linewidth, patternnum=1;
    array double x[numpoints], y[numpoints];
    class CPlot plot;
    double begin;

    begin = 0.01;
    for(j=0; j< M; j++) {
      for(i=0; i<N; i++) {
        lindata(begin, begin+180-0.01, x);
        y = sin(x*M_PI/180);
        for(k=0; k<numpoints; k++) {
          y[k] += sign(y[k])*1;
        }
        plot.data2D(x, y);
        linetype = i+1;
        linewidth = 2;
        plot.plotType(PLOT_PLOTTYPE_BOXES, i+j*N, linetype, linewidth);
        plot.boxWidth(30);
        if(i==1)
           plot.boxFill(i+j*N, PLOT_BOXFILL_SOLID);
        else if(i==2)  {
           plot.boxFill(i+j*N, PLOT_BOXFILL_SOLID, 0.25);
           plot.boxBorder(i+j*N, 1);
        }
        else if(i==3)  {
           plot.boxBorder(i+j*N);
           plot.boxFill(i+j*N, PLOT_BOXFILL_PATTERN, patternnum);
        }
        begin += 180;
      }
    }
    plot.axisRange(PLOT_AXIS_X, -60, N*M*180+60);
    plot.ticsRange(PLOT_AXIS_X, 180, 0, N*M*180);
    plot.axisRange(PLOT_AXIS_Y, -2.5, 2.5);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::**boxBorder**(), **CPlot**::**boxWidth**(), **CPlot**::**plotType**().

CPlot::boxWidth

Synopsis
#include <**chplot.h**>
void boxWidth(**double** *width*);

Syntax
boxWidth(*width*)

Purpose
Set the width for a box.

Return Value
None.

Parameters
width The width of boxes to be drawon.

Description
This function specifies the width of boxes for a plot type of **PLOT_PLOTTYPE_BOXES**, **PLOT_PLOTTYPE_BOXERRORBARS**,

PLOT_PLOTTYPE_BOXXYERRORBARS, and **PLOT_PLOTTYPE_CANDLESTICKS** specified by member function **CPlot**::**plotType**().

By default, adjacent boxes are extended in width until they touch each other. A different default width may be specified using the this member function.

The parameter `width` for the boxwidth is interpreted as being a number of units along the current x axis. If the x axis is a log-scale then the value of boxwidth is absolute only at x=1; this physical width is maintained everywhere along the axis (i.e. the boxes do not become narrower the value of x increases). If the range spanned by a log scale x axis is far from x=1, some experimentation may be required to find a useful value of boxwidth.

The default is superseded by explicit width information taken from an extra data column for a plot type of **PLOT_PLOTTYPE_BOXES**, **PLOT_PLOTTYPE_BOXERRORBARS**, or **PLOT_PLOTTYPE_BOXXYERRORBARS**, In a four-column data set, the fourth column will be interpreted as the box width unless the `width` is set to -2.0, in which case the width will be calculated automatically.

To set the box width to automatic for four-column data, use

```
plot.boxwidth(-2);
```

The same effect can be achieved with the option of `using` keyword for member function **CPlot**::**dataFile**() as follows.

```
plot.dataFile(datafile, "using 1:2:3:4:(-2)");
```

To set the box width to an absolute value of 30, use

```
plot.boxWidth(2);
```

Example

See an example on page 30 for **CPlot**:**boxFill**().

See Also

CPlot::**boxBorder**(), **CPlot**::**boxFill**(), **CPlot**::**plotType**().

CPlot::changeViewAngle

Synopsis

#include <**chplot.h**>

void changeViewAngle(**double** *x_rot*, **double** *z_rot*);

Purpose

Change the viewpoint for a 3D plot.

Return Value

None.

Parameters

x_rot The angle of rotation for the plot (in degrees) along an axis horizontal axis of the screen.

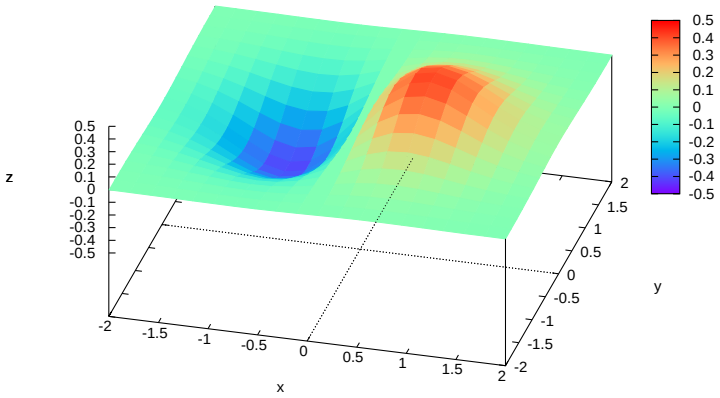
z_rot The angle of rotation for the plot (in degrees) along an axis perpendicular to the screen.

Description
This function provides rotation of a 3D plot. *x_rot* and *z_rot* are bounded by the range [0:180] and the range [0:360] respectively. By default, 3D plots are displayed with $x\_rot = 60°$ and $z\_rot = 30°$.

Example
Compare with the output for examples in **CPlot**::**data3D**() and **CPlot**::**data3DSurface**().

```
/* File: changeViewAngle.ch */
#include <math.h>
#include <chplot.h>

int main() {
    array double x[20], y[20], z[400];
    int i, j;
    class CPlot plot;

    lindata(-2, 2, x);
    lindata(-2, 2, y);
    for (i=0; i<20; i++) {
        for(j=0; j<20; j++) {
            z[i*20+j] = x[i]*exp(-x[i]*x[i]-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.changeViewAngle(45, 15);
    plot.plotting();
}
```

Output

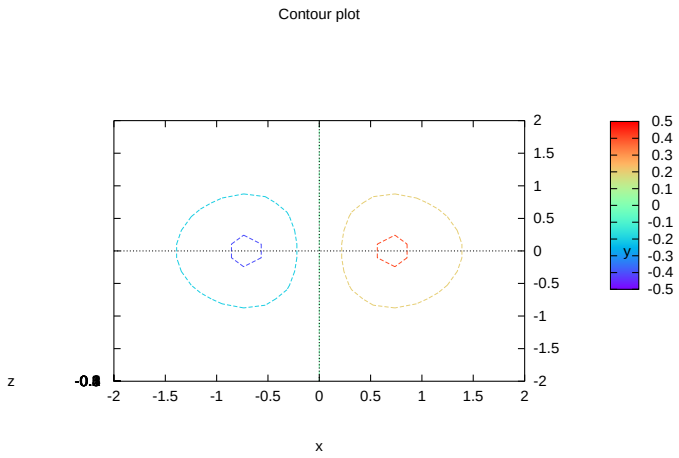


Example

```
/* File: changeViewAngle_2.ch */
#nclude <chplot.h>
```

```
#include <math.h>

int main() {
    array double x[20], y[20], z[400];
    int datasetnum =0, i, j;
    class CPlot plot;

    lindata(-2, 2, x);
    lindata(-2, 2, y);
    for (i=0; i<20; i++) {
        for(j=0; j<20; j++) {
            z[i*20+j] = x[i]*exp(-x[i]*x[i]-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.changeViewAngle(0,0);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.contourMode(PLOT_CONTOUR_BASE);
    plot.showMesh(PLOT_OFF);
    plot.title("Contour plot");
    plot.plotting();
}
```

Output



See Also

CPlot::data3D(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**().

CPlot::circle

Synopsis

#include <chplot.h>

int circle(double *x_center*, **double** *y_center*, **double** *r*);

Purpose

Add a circle to a 2D plot.

Return Value

This function returns 0 on success and -1 on failure.

Parameters

x_center The x coordinate of the center of the circle.

y_center The y coordinate of the center of the circle.

r The radius of the circle.

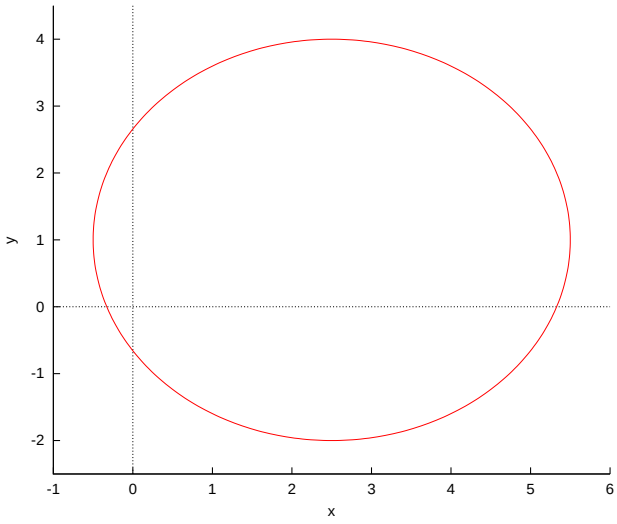
Description

This function adds a circle to a 2D plot. It is a convenience function for creation of geometric primitives. A circle added with this function is counted as a data set for later calls to **CPlot**::**legend**() and **CPlot**::**plotType**(). For rectangular plots, *x* and *y* are the coordinates of the center of the circle and *r* is the radius of the circle, all specified in units of the x and y axes. For polar plots, the location of the center of the circle is specified in polar coordinates where *x* is θ and *y* is r.

Example 1

```
/* File: circle.ch */
#include <chplot.h>

int main() {
    double x_center = 2.5, y_center = 1.0, r = 3;
    class CPlot plot;

    plot.circle(x_center, y_center, r);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_Y, -2.5, 4.5);
    plot.plotting();
}
```

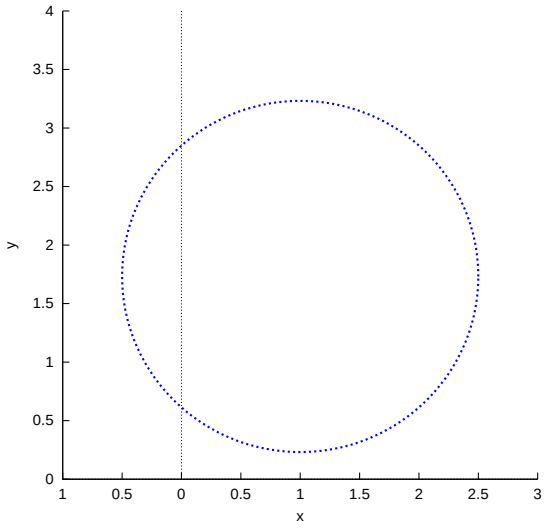
Output



Example 2

```
/* File: circle_2.ch */
#include <chplot.h>
#include <math.h>

int main() {
    double x_center = M_PI/3, y_center = 2.0, r = 1.5;
    class CPlot plot;

    plot.polarPlot(PLOT_ANGLE_RAD); /* Polar coordinate system */
    plot.circle(x_center, y_center, r);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.lineType(0, 3, 4);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_X, -1, 3);
    plot.axisRange(PLOT_AXIS_Y, 0, 4);
    plot.plotting();
}
```

Output



See Also

CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**line**(), **CPlot**::**outputType**(), **CPlot**::**plotType**(), **CPlot**::**point**(), **CPlot**::**polygon**(), **CPlot**::**rectangle**().

CPlot::colorBox

Synopsis

#include <**chplot.h**>

void colorBox(**int** *flag*);

Purpose

Enable or disable the drawing of a color box for a 3D surface plot.

Return Value
None.

Parameter
flag This parameter can be set to:

PLOT_ON Enable the color box.

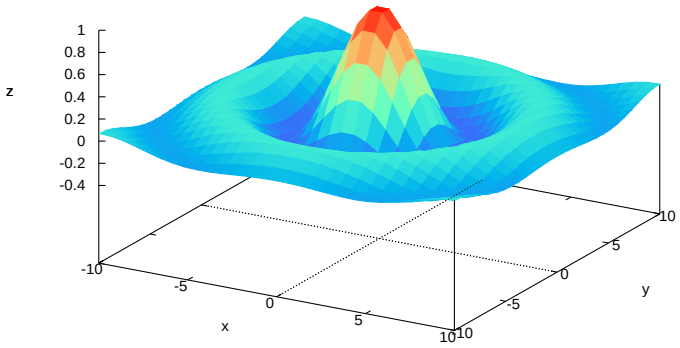
PLOT_OFF Disable the color box.

Description
Enable or disable the drawing of a color box, which contains the color scheme, i.e. the gradient of the smooth color with the maximum and minimum values of the color palette. This is applicable only for 3D surface plots. By default, the color box is drawn.

Example 1
Compare with the output for the example in **CPlot**::**data3D**().

```
/* File: colorBox.ch */
#include <chplot.h>
#include <math.h>
#include <numeric.h>

int main() {
    double x[30], y[30], z[900];
    double r;
    int i, j;
    class CPlot plot;

    lindata(-10, 10, x);
    lindata(-10, 10, y);
    for(i=0; i<30; i++) {
        for(j=0; j<30; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[30*i+j] = sin(r)/r;
        }
    }
    plot.data3D(x, y, z);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
}
```

Output



See Also
CPlot::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**contourMode**(), **CPlot**::**plotType**().

CPlot::contourLabel

Synopsis
#include <**chplot.h**>
void contourLabel(**int** *flag*);

Purpose
Set display of contour line elevation labels for 3D plots on or off.

Return Value
None.

Parameter
flag Enable or disable drawing of contour line labels.

PLOT_ON Enable contour labels.

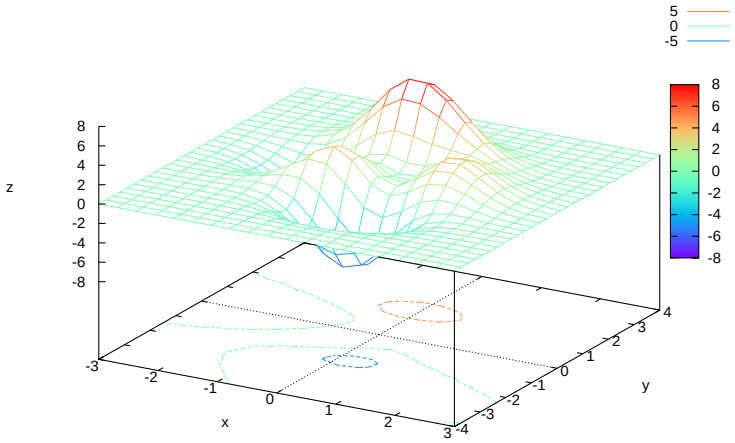
PLOT_OFF Disable contour labels.

Description
Enable or disable contour line labels for 3D surface plots. labels appear with the plot legend. By default, labels for contours are not displayed.

Example 1
Compare with the output for examples in **CPlot**::**data3D**(), **CPlot**::**data3DSurface**(). **CPlot**::**contourLevels**(), and **CPlot**::**showMesh**().

```
/* File: contourLabel.ch */
#include <math.h>
#include <chplot.h>

int main() {
    double x[20], y[30], z[600];
    int datasetnum =0, i, j;
    class CPlot plot;

    lindata(-3, 3, x);
    lindata(-4, 4, y);
    for(i=0; i<20; i++) {
        for(j=0; j<30; j++) {
            z[30*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.contourLabel(PLOT_ON);
    plot.contourMode(PLOT_CONTOUR_BASE);
    plot.plotting();
}
```

Output

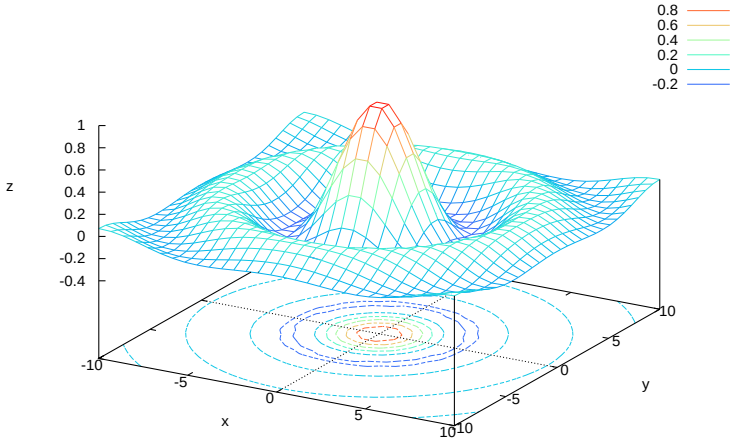


Example 2

```
/* File: contourLabel_2.ch */
#include <math.h>
#include <chplot.h>

int main() {
    double x[30], y[30], z[900];
    double r;
    int datasetnum =0, i, j;
    class CPlot plot;

    lindata(-10, 10, x);
```

```
    lindata(-10, 10, y);
    for(i=0; i<30; i++) {
        for(j=0; j<30; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[30*i+j] = sin(r)/r;
        }
    }
    plot.data3D(x, y, z);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.contourLabel(PLOT_ON);
    plot.contourMode(PLOT_CONTOUR_BASE);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
}
```

Output



See Also

CPlot::**data3D**(), **CPlot**::**contourLevels**(), **CPlot**::**contourMode**(), **CPlot**::**legend**(), **CPlot**::**showMesh**().

CPlot::contourLevels

Synopsis

#include <**chplot.h**>

void contourLevels(double *levels*[:], ... /* [**int** *n*] */);

Syntax

contourLevels(*level*)

contourLevels(*level*, *n*)

Purpose

Set contour levels for 3D plot to be displayed at specific locations.

Return Value
None.

Parameter
levels An array of z-axis levels for contours to be drawn.

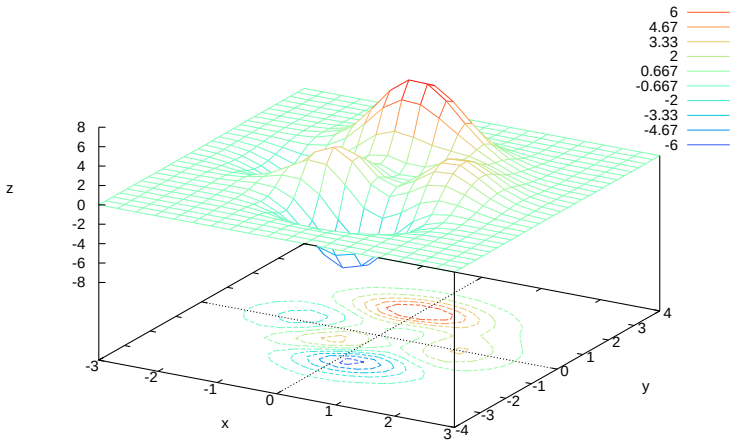
n The number of elements of array *levels*.

Description
This function allows contour levels for 3D grid data to be displayed at any desired z-axis position. The contour levels are stored in an array where the lowest contour is in the first array element.

Example 1
Compare with the output from the example in **CPlot**::**contourLabel**().

```
/* File: contourLevels.ch */
#include <math.h>
#include <chplot.h>

int main() {
    float x[20], y[30], z[600];
    double levels[10];
    int datasetnum =0, i, j;
    class CPlot plot;

    lindata(-6, 6, levels);
    lindata(-3, 3, x);
    lindata(-4, 4, y);
    for(i=0; i<20; i++) {
        for(j=0; j<30; j++) {
            z[30*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.contourLabel(PLOT_ON);
    plot.contourMode(PLOT_CONTOUR_BASE);
    plot.contourLevels(levels);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
}
```

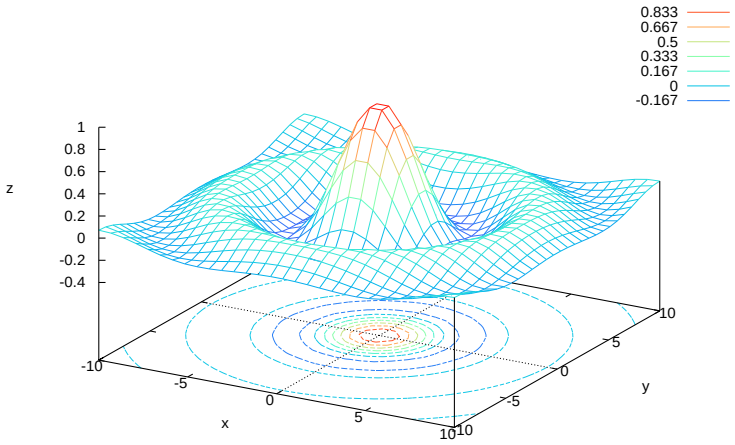
Output



Example 2

```
/* File: contourLevels_2.ch */
#include <math.h>
#include <chplot.h>

int main() {
    double x[30], y[30], z[900];
    double levels[6];
    double r;
    int datasetnum =0, i, j;
    class CPlot plot;

    lindata(-10, 10, x);
    lindata(-10, 10, y);
    lindata(-0.2, 0.8, levels);
    for(i=0; i<30; i++) {
        for(j=0; j<30; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[30*i+j] = sin(r)/r;
        }
    }
    plot.data3D(x, y, z);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.contourLabel(PLOT_ON);
    plot.contourMode(PLOT_CONTOUR_BASE);
    plot.contourLevels(levels);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
}
```

Output



See Also
CPlot::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**contourLabel**(), **CPlot**::**contourMode**(), **CPlot**::**showMesh**().

CPlot::contourMode

Synopsis
#include <**chplot.h**>
void contourMode(int *mode*);

Purpose
Selects options for the drawing of contour lines in 3D plots.

Return Value
None.

Parameter
mode The following contour modes are available and can be combined using the logical or (|) operator:

PLOT_CONTOUR_BASE Contour lines drawn on the xy plane.

PLOT_CONTOUR_SURFACE Contour lines drawn on the surface of the plot.

Description
This option is available for display of 3D grid data. For the plot type of `PLOT_PLOTTYPE_LINES`, the positions of the contour levels are determined internally unless explicitly set using **CPlot**::**contourLevels**(). The **PLOT_CONTOUR_SURFACE** option does not work with hidden line removal. The hidden lines are removed by default. It can be disabled by member function **CPlot**::**removeHiddenLine**().

For the plot type of PLOT_PLOTTYPE_SURFACES, the contour mode of **PLOT_CONTOUR_SURFACE** will add a projected map on the xy plane. the contour mode of **PLOT_CONTOUR_SURFACE** will have a projected map on the xy plane only.

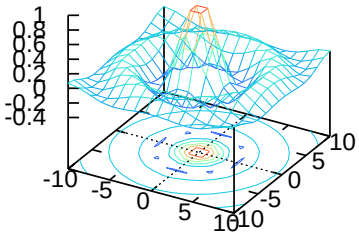
Example

```
/* File: contourMode.ch */
#include <chplot.h>
#include <math.h>

int main() {
    double x[16], y[16], z[256];
    double r;
    int i, j;
    class CPlot subplot, *spl;

    lindata(-10, 10, x);
    lindata(-10, 10, y);
    for(i=0; i<16; i++) {
        for(j=0; j<16; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[16*i+j] = sin(r)/r;
        }
    }
    subplot.subplot(2,3);
    spl = subplot.getSubplot(0,0);
    spl->data3D(x, y, z);
    spl->plotType(PLOT_PLOTTYPE_LINES, 0);
    spl->contourMode(PLOT_CONTOUR_BASE);
    spl->removeHiddenLine(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->colorBox(PLOT_OFF);
    spl->title("PLOT_CONTOUR_BASE");

    spl = subplot.getSubplot(0,1);
    spl->data3D(x, y, z);
    spl->plotType(PLOT_PLOTTYPE_LINES, 0);
    spl->contourMode(PLOT_CONTOUR_SURFACE);
    spl->removeHiddenLine(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->colorBox(PLOT_OFF);
    spl->title("PLOT_CONTOUR_SURFACE");

    spl = subplot.getSubplot(0,2);
    spl->data3D(x, y, z);
    spl->plotType(PLOT_PLOTTYPE_LINES, 0);
    spl->contourMode(PLOT_CONTOUR_BASE|PLOT_CONTOUR_SURFACE);
    spl->removeHiddenLine(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->colorBox(PLOT_OFF);
    spl->title("PLOT_CONTOUR_BASE|PLOT_CONTOUR_SURFACE");

    spl = subplot.getSubplot(1,0);
    spl->data3D(x, y, z);
    spl->contourMode(PLOT_CONTOUR_SURFACE);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->colorBox(PLOT_OFF);
    spl->title("PLOT_CONTOUR_SURFACE");
```

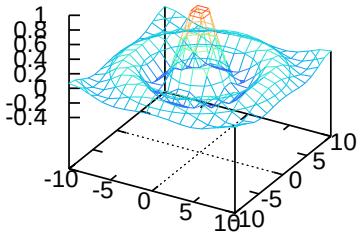
```
    spl = subplot.getSubplot(1,1);
    spl->data3D(x, y, z);
    spl->contourMode(PLOT_CONTOUR_BASE);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->colorBox(PLOT_OFF);
    spl->title("PLOT_CONTOUR_BASE");
    subplot.plotting();
}
```

Output

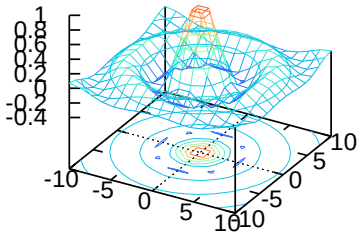
PLOT_CONTOUR_BASE



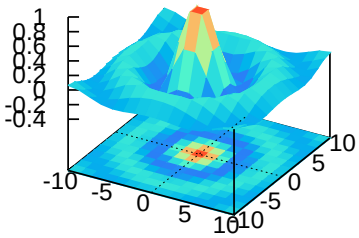
PLOT_CONTOUR_SURFACE



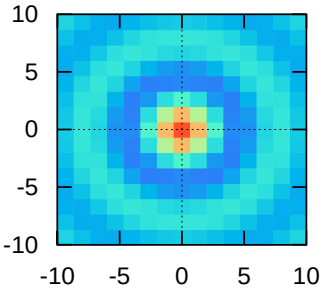
PLOT_CONTOUR_BASE|PLOT_CONTOUR_SURFACE



PLOT_CONTOUR_SURFACE



PLOT_CONTOUR_BASE



See Also

CPlot::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**contourLevels**(), **CPlot**::**removeHiddenLine**(), **CPlot**::**showMesh**().

CPlot::coordSystem

Synopsis

#include <**chplot.h**>

void coordSystem(**int** *coord_system*, ... /* [**int** *angle_unit*] */);

Syntax

coordSystem(*coord_system*)
coordSystem(*coord_system*, *angle_unit*)

Purpose
Select coordinate system for 3D plots.

Return Value
None.

Parameters
coord_system The coordinate system can be set to any of the following:

PLOT_COORD_CARTESIAN Cartesian coordinate system. Each data point consists of three values (x,y,z).

PLOT_COORD_SPHERICAL Spherical coordinate system. Each data point consists of three values (θ,ϕ,r).

PLOT_COORD_CYLINDRICAL Cylindrical coordinates. Each data point consists of three values (θ,z,r).

angle_unit an optional parameter to specify the unit for mesurement of an angular position in **PLOT_COORD_SPHERICAL** and **PLOT_COORD_CYLINDRICAL** coordinate systems. The following options are available:

PLOT_ANGLE_DEG Angles measured in degree.

PLOT_ANGLE_RAD Angles measured in radian.

Description
This function selects the coordinate system for 3D plots. For a spherical coordinate system, points are mapped to Cartesian space by:

$$\begin{aligned} x &= r\cos(\theta)\cos(\phi) \\ y &= r\sin(\theta)\cos(\phi) \\ z &= r\sin(\phi) \end{aligned}$$

For a cylindrical coordinate system, points are mapped to Cartesian space by:

$$\begin{aligned} x &= r\cos(\theta) \\ y &= r\sin(\theta) \\ z &= z \end{aligned}$$

The default coordinate system is **PLOT_COORD_CARTESIAN**. For **PLOT_COORD_SPHERICAL** and **PLOT_COORD_CYLINDRICAL**, the default unit for *angle_unit* is **PLOT_ANGLE_RAD**.

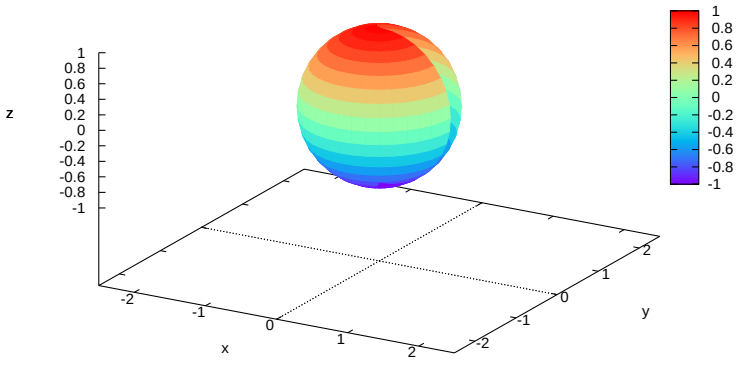
Example 1

See **CPlot**::**data3D**().

Example 2

```
/* File: coordSystem2.ch */
#include <chplot.h>
#include <math.h>

int main() {
    array double theta[37], phi[19], r[703];
    class CPlot plot;

    lindata(0, 2*M_PI, theta);
    lindata(-M_PI/2, M_PI/2, phi);
    r = (array double [703])1;
    plot.data3D(theta, phi, r);
    plot.coordSystem(PLOT_COORD_SPHERICAL);
    plot.axisRange(PLOT_AXIS_XY, -2.5, 2.5);
    plot.plotting();
}
```

Output

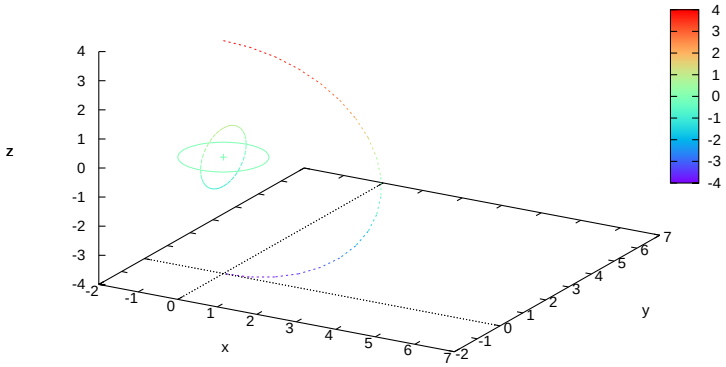


Example 3

```
/* File: coordSystem3.ch */
#include <chplot.h>
#include <math.h>

int main() {
    array double theta1[37], phi1[37], r1[37];
    array double theta2[37], phi2[37], r2[37];
    array double theta3[20], phi3[20], r3[20];
    class CPlot plot;

    lindata(0, 2*M_PI, theta1);
    phi1 = (array double [37])0;
    r1 = (array double [37])1;
```

```
    theta2 = (array double [37])M_PI/2;
    lindata(phi2, 0, 2*M_PI, phi2);
    r2 = (array double [37])1;
    theta3 =  (array double [20])0;
    lindata(-M_PI/2, M_PI/2, phi3);
    r3 = (array double [20])4;
    plot.data3D(theta1, phi1, r1);
    plot.data3D(theta2, phi2, r2);
    plot.data3D(theta3, phi3, r3);
    plot.point(0, 0, 0);
    plot.coordSystem(PLOT_COORD_SPHERICAL);
    plot.axisRange(PLOT_AXIS_XY, -2, 7);
    plot.ticsLevel(0);
    plot.removeHiddenLine(PLOT_OFF);
    plot.plotting();
}
```

Output

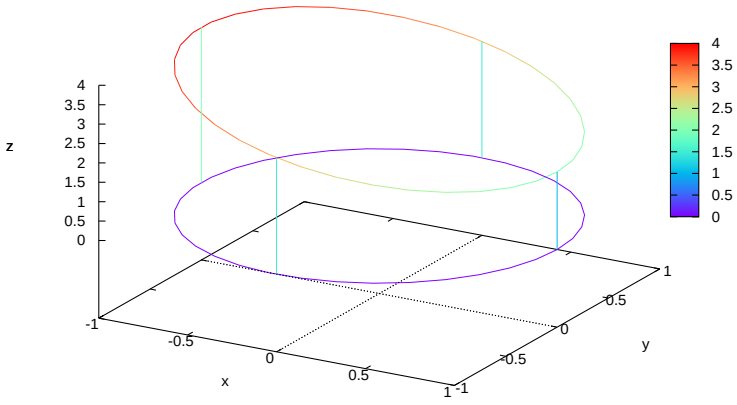


Example 4

```
/* File: coordSystem4.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double r1[numpoints], theta1[numpoints], z1[numpoints];
    array double r2[numpoints], theta2[numpoints], z2[numpoints];
    class CPlot plot;

    lindata(0, 360, theta1);
    z1=(array double [numpoints])3+
       sin((theta1-(array double [numpoints])90)*M_PI/180);
    r1=(array double [numpoints])1;
    lindata(0, 360, theta2);
    z2=(array double [numpoints])0;
    r2=(array double [numpoints])1;
    plot.data3D(theta1, z1, r1);
    plot.data3D(theta2, z2, r2);
```

```
    plot.coordSystem(PLOT_COORD_CYLINDRICAL, PLOT_ANGLE_DEG);
    plot.line(0, 0, 1, 0, 2, 1);
    plot.line(90, 0, 1, 90, 3, 1);
    plot.line(180, 0, 1, 180, 4, 1);
    plot.line(270, 0, 1, 270, 3, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.lineType(0, 1, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 1);
    plot.lineType(1, 1, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 2);
    plot.lineType(2, 1, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 3);
    plot.lineType(3, 1, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 4);
    plot.lineType(4, 1, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 5);
    plot.lineType(5, 1, 1);
    plot.plotting();
}
```

Output

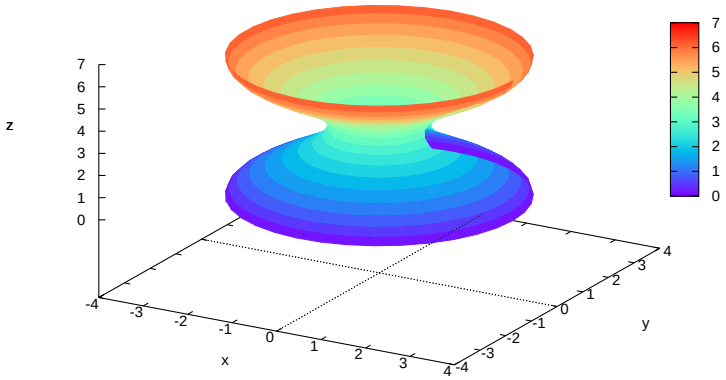


Example 5

```
/* File: coordSystem5.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double theta[36], z[20], r[720];
    int i, j;
    class CPlot plot;

    lindata(0, 360, tetha);
    lindata(0, 2*M_PI, z);
    for(i=0; i<36; i++) {
        for(j=0; j<20; j++) {
            r[i*20+j] = 2+cos(z[j]);
        }
```

```
    }
    plot.data3D(theta, z, r);
    plot.coordSystem(PLOT_COORD_CYLINDRICAL, PLOT_ANGLE_DEG);
    plot.axisRange(PLOT_AXIS_XY, -4, 4);
    plot.plotting();
}
```

Output



See Also
CPlot::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**().

CPlot::data

Synopsis
#include <**chplot.h**>
int data(void **x*, **int** *row*, **int** `col`);

Purpose
Add 2D, 3D, or multi-dimensional data set to an instance of the **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x A two-dimensional array of double type used for the plot.

row The number of rows for the array `x`.

col The number of columns for the array `x`.

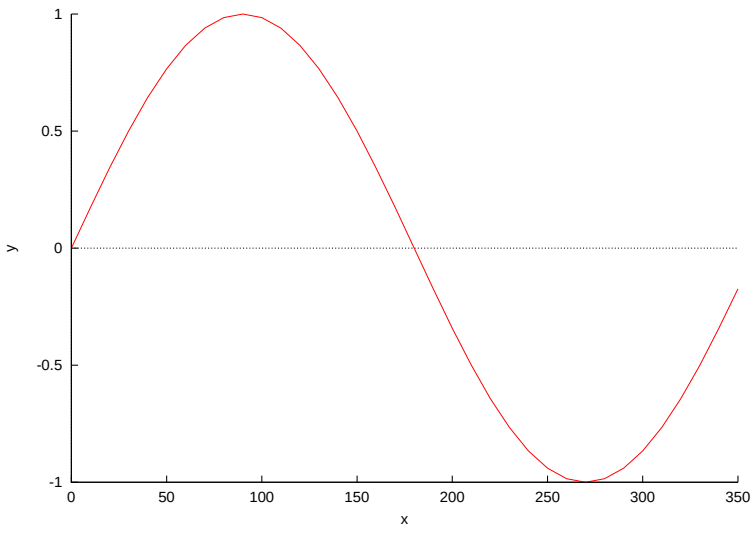
Description

The data for a plot can be placed in either a file or in the memory. The opaque pointer `x` is the address for a two-dimensional array of double type. The size of the array is specified by its number of rows and columns. The data with multiple columns for a plot type such as candlesticks, finance bars, boxes, etc. can be added by this member function.

Example 1
In this example, a data set for 2D plot is added.

```
/* File: data.ch */
#include <math.h>
#include <chplot.h>

#define ROW 36
#define COL 2
int main() {
    int i;
    double a[ROW][COL];
    class CPlot plot;

    for(i=0; i< ROW; i++) {
      a[i][0] = i*10;
      a[i][1] = sin(a[i][0]*M_PI/180);
    }
    plot.data(a, ROW, COL);
    plot.plotting();
    return 0;
}
```

Output

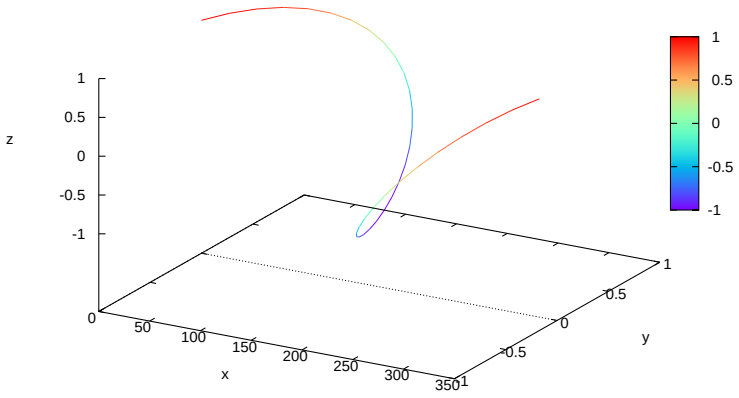


Example 2 In this example, a data set for 3D plot is added.

```
/* File: data_1.ch */
#include <math.h>
#include <chplot.h>

#define ROW 36
```

```
#define COL 3
int main() {
    int i;
    double a[ROW][COL];
    class CPlot plot;

    for(i=0; i< ROW; i++) {
      a[i][0] = i*10;
      a[i][1] = sin(a[i][0]*M_PI/180);
      a[i][2] = cos(a[i][0]*M_PI/180);
    }
    plot.data(a, ROW, COL);
    plot.dimension(3);
    plot.plotting();
    return 0;
}
```

Output



Example 3 In this example, a data set for candlesticks is added by `plot.data(a, ROW, COL)` and `plot.data(b, ROW, COL)`.

```
/* File: data_2.ch */
/* File: data_2.cpp to process data in candlesticks.dat */
/*       compare with plotType_cs.ch */
#include <chplot.h>

#define ROW 10
#define COL 5
int main() {
    class CPlot plot;
    double a[ROW][COL], b[ROW][COL];
    char filename[]="candlesticks.dat";
    FILE *stream;
    int i;

    stream = fopen(filename, "r");
    if(stream == NULL) {
      fprintf(stderr, "Error: cannot open file '%s' for reading\n", filename);
      return -1;
```

```
    }
    /* "using 1:3:2:6:5" */
    for(i = 0; i<ROW; i++) {
      fscanf(stream, "%lf%lf%lf%*lf%lf%lf",
            &a[i][0], &a[i][2], &a[i][1], &a[i][4], &a[i][3]);
      printf("%lf %lf %lf %lf %lf\n",
            a[i][0], a[i][1], a[i][2], a[i][3], &a[i][4]);
    }
    rewind(stream);
    /* using 1:4:4:4:4" */
    for(i = 0; i<ROW; i++) {
         fscanf(stream, "%lf%*lf%*lf%lf%*lf%*lf", &b[i][0], &b[i][1]);
         b[i][2] = b[i][1];
         b[i][3] = b[i][1];
         b[i][4] = b[i][1];
         printf("%lf %lf\n", b[i][0], b[i][1]);
    }
    fclose(stream);

    plot.label(PLOT_AXIS_X, "");
    plot.label(PLOT_AXIS_Y, "");
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot.title("box-and-whisker plot adding median value as bar");
    plot.axisRange(PLOT_AXIS_X, 0, 11);
    plot.axisRange(PLOT_AXIS_Y, 0, 10);
    //plot.dataFile("candlesticks.dat", "using 1:3:2:6:5");
    //plot.legend("'candlesticks.dat' using 1:3:2:6:5", 0);
    plot.data(a, ROW, COL);
    plot.legend("array a", 0);
    plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 0, "linetype 1 linewidth 2 whiskerbars");
    plot.boxFill(0, PLOT_BOXFILL_EMPTY);
    //plot.dataFile("candlesticks.dat", "using 1:4:4:4:4");
    //plot.legend("'candlesticks.dat' using 1:4:4:4:4", 1);
    plot.data(b, ROW, COL);
    plot.legend("array b", 1);
    plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 1, "linetype -1 linewidth 2");
    plot.boxWidth(0.2);
    plot.plotting();
    return 0;
}
```

Output

The output is the same as that from program plotType_cs.ch on page 151 for **CPlot**::**plotType**().

See Also

CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**dataFile**(), **CPlot**::**plotType**().

CPlot::data2D

Synopsis

#include <**chplot.h**>

int data2D(**array double** *x*[&], **array double** &*y*);

Purpose
Add one or more 2D data sets to an instance of **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

x a one-dimensional array of reference type used for the x axis of the plot.

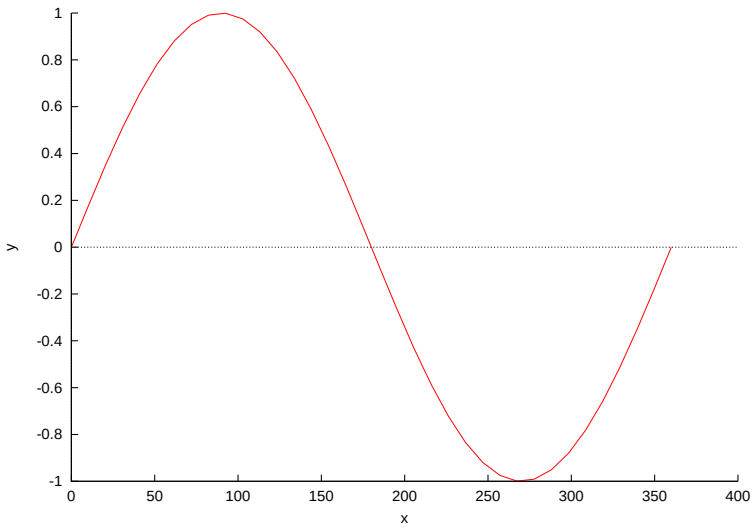
y an array of reference type. If the size of array x is n, array y is a one-dimensional array of size n or a two-dimensional array of size m x n containing m curves, each of which is plotted against *x*.

Description
This function adds the specified x-y data to a previously declared instance of the **CPlot** class. The parameter *x* is a one-dimensional array of size n. The parameter *y* is a one-dimensional array of size n or a two-dimensional array of size m x n. In the case that *y* is m x n, each of the m rows of *y* is plotted against *x*. Each of the rows of y is a separate data set. The *x* and *y* arrays can be of any supported data type. Conversion of the data to **double** type is performed internally. Data points with a y value of NaN are internally removed before plotting occurs. ”Holes” in a data set can be constructed by manually setting elements of *y* to this value. The plot of the data is generated using the **CPlot**::**plotting** member function.

Example 1
Compare with the output for the examples in **CPlot**::**arrow**(), **CPlot**::**autoScale**(), **CPlot**::**borderOffsets**(), **CPlot**::**displayTime**(), **CPlot**::**label**(), **CPlot**::**ticsLabel**(), **CPlot**::**margins**(), **CPlot**::**ticsDirection**(), **CPlot**::**ticsFormat**(), **CPlot**::**ticsLocation**(), and **CPlot**::**title**().

```
/* File: data2D.ch */
#include <math.h>
#include <chplot.h>
#include <numeric.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.data2D(x, y);
    plot.plotting();
}
```

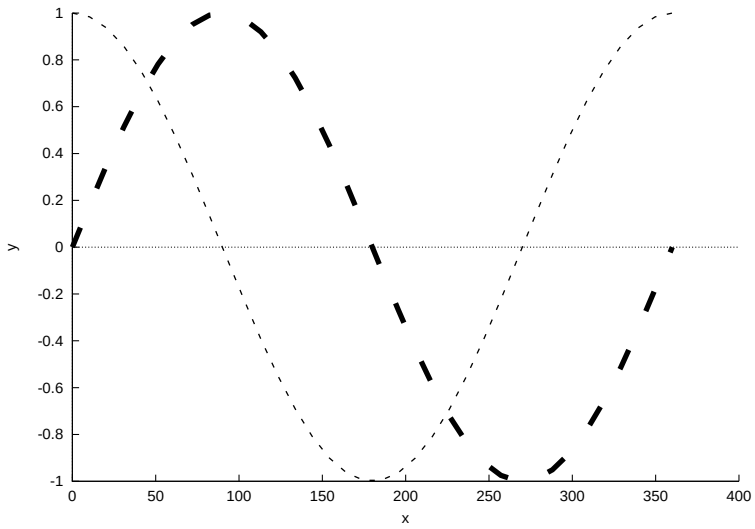
Output



Example 2

```
/* File: data2D_2.ch */
#include <math.h>
#include <chplot.h>
#include <numeric.h>
#define CURVE1 0
#define CURVE2 1

int main() {
    int numpoints = 36;
    array double x[numpoints], y[2][numpoints];
    int i;
    class CPlot plot;

    lindata(0, 360, x);
    for(i=0; i<numpoints; i++) {
        y[0][i] = sin(x[i]*M_PI/180);
        y[1][i] = cos(x[i]*M_PI/180);
    }
    plot.data2D(x, y);
    plot.plotType(PLOT_PLOTTYPE_LINES, CURVE1);
    plot.lineType(CURVE1, 1, 20); /* set first data set to use the default line
                                     type and a width of twenty  times the
                                      default width */
    plot.plotType(PLOT_PLOTTYPE_LINES, CURVE2);
    plot.lineType(CURVE2, 1, 5); /* set second data set to use the default line
                                    type and a width of five times the default */
    plot.plotting();
}
```

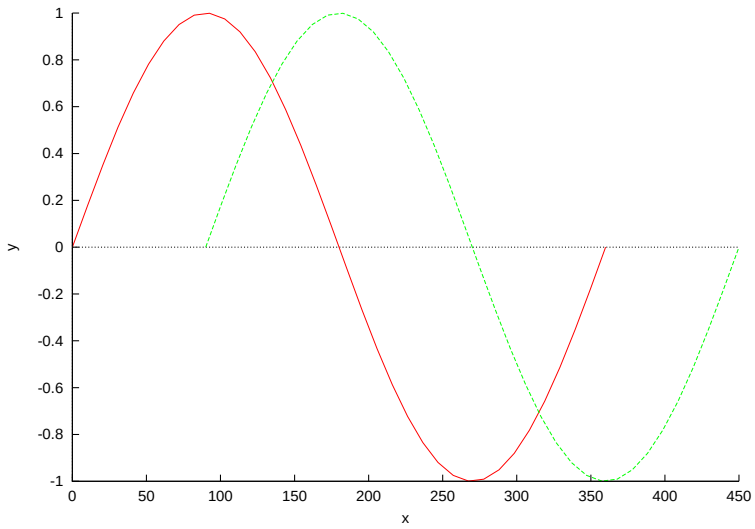
Output



Example 3

```
/* File: data2D_3.ch */
/* Two curves have a phase shift */
#include <math.h>
#include <chplot.h>
#include <numeric.h>

int main() {
    int numpoints = 36;
    array double x1[numpoints], y1[numpoints];
    array double x2[numpoints];
    class CPlot plot;

    lindata(0, 360, x1);
    y1 = sin(x1*M_PI/180);
    lindata(90, 450, x2);
    plot.data2D(x1, y1);
    plot.data2D(x2, y1);
    plot.plotting();
}
```

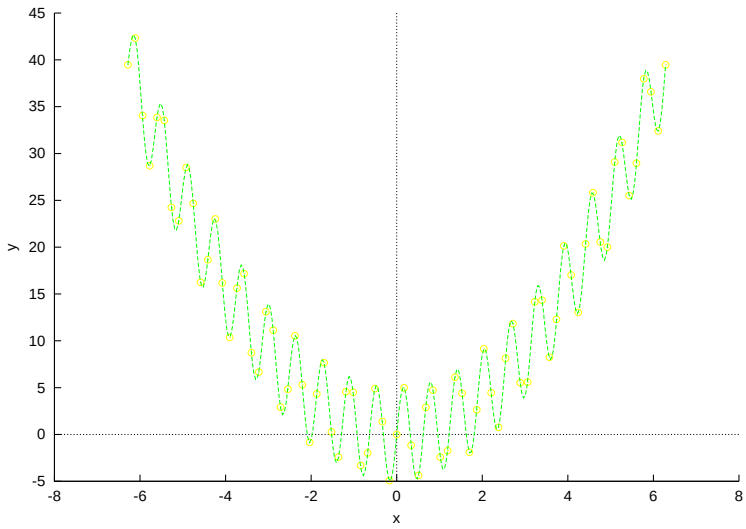
Output



Example 4

```
/* File: data2D_4.ch */
#include <chplot.h>
#include <math.h>
#include <numeric.h>

int main() {
    array float x1[75], y1[75];
    array float x2[300], y2[300];
    class CPlot plot;

    lindata(-2*M_PI, 2*M_PI, x1);
    lindata(-2*M_PI, 2*M_PI, x2);
    y1 = x1.*x1+5*sin(10*x1);
    y2 = x2.*x2+5*sin(10*x2);
    plot.data2D(x1, y1);
    plot.data2D(x2, y2);
    plot.plotType(PLOT_PLOTTYPE_POINTS, 0);
    plot.pointType(0, 6, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 1);
    plot.lineType(1, 2, 1);
    plot.plotting();
}
```

Output



See Also
CPlot::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**dataFile**(), **CPlot**::**plotting**(), **plotxy**().

CPlot::data2DCurve

Synopsis
#include <**chplot.h**>
int data2DCurve(**double** *x*[], **double** *y*[], int n);

Purpose
Add a set of data for 2D curve to an instance of **CPlot** class.
Return Value
This function returns 0 on success and -1 on failure.

Parameters
x a one-dimensional array of double type used for the x axis of the plot.

y a one-dimentaional array of double type for the y axis.

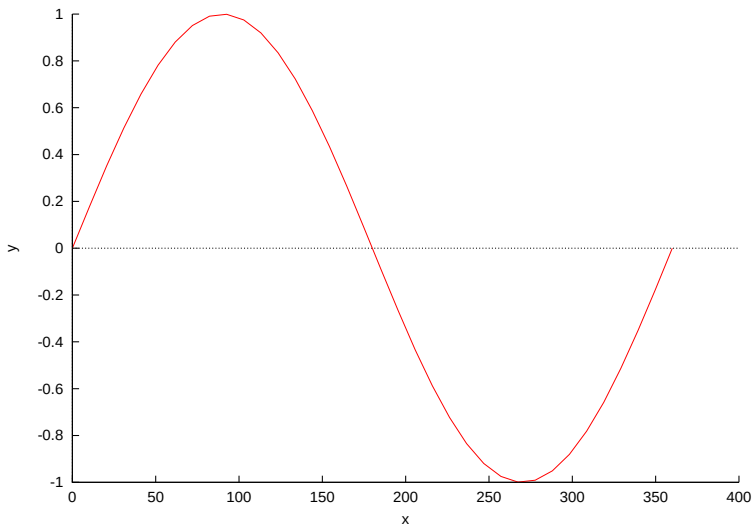
n number of elements of arrays *x* and *y*.

Description
This function adds the specified x-y data to a previously declared instance of the **CPlot** class. The parameter *x* is a one-dimensional array of size n. The parameter *y* is a one-dimensional array of size n. Data points with a y value of NaN are internally removed before plotting occurs. ”Holes” in a data set can be constructed by manually setting elements of *y* to this value. The plot of the data is generated using the **CPlot**::**plotting** member function.

Example 1
Compare with the output for examples in **CPlot**::**arrow**(), **CPlot**::**autoScale**(), **CPlot**::**borderOffsets**(), **CPlot**::**data2D**(), **CPlot**::**displayTime**(), **CPlot**::**label**(), **CPlot**::**ticsLabel**(), **CPlot**::**margins**(), **CPlot**::**ticsDirection**(), **CPlot**::**ticsFormat**(), **CPlot**::**ticsLocation**(), and **CPlot**::**title**().

```
/* File: data2DCurve.ch */
#include <math.h>
#include <chplot.h>
#ifndef M_PI
#define M_PI            3.14159265358979323846
#endif

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i< NUM; i++) {
      x[i] = i*10;
      y[i] = sin(x[i]*M_PI/180);
    }
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::**data2D**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**dataFile**(), **CPlot**::**plotting**(), **plotxy**().

CPlot::data3D

Synopsis
#include <chplot.h>

int data3D(**array double** *x*[&], **array double** *y*[&], array **double** &*z*);

Purpose
Add one or more 3D data sets to an instance of **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x a one-dimensional array of size n_x used for the x axis of the plot.

y a one-dimensional array of size n_y used for the y axis of the plot.

z a one-dimensional array of size n_z, or a two dimensional array of size m x n_z, containing m 3D data sets.

Description
This function adds one or more 3D data sets to an existing plot variable. For Cartesian data, *x* is a one-dimensional array of size n_x and *y* is a one-dimensional array of size n_y. *z* can be of two different sizes depending on what type of data is to be plotted. If the data are for a 3D curve, *z* is a one-dimensional array of size n_z or a two-dimensional array of size m x n_z, with $n_x = n_y = n_z$. If the data are for a 3D surface or grid, *z* is m x n_z, with $n_z = n_x \cdot n_y$. For cylindrical or spherical data *x* is a one dimensional array of size n_x (representing θ), *y* is a one dimensional array of size n_y (representing z or ϕ), and *z* is of the size m x n_z(representing r) where $n_x = n_y = n_z$. Each of the m rows of *z* are plotted against x and y, and correspond to a separate data set. In all cases these data arrays can be of any supported data type. Conversion of the data to **double** type is performed internally. For grid data, hidden line removal is enabled automatically (see **CPlot**::**removeHiddenLine**()). If it is desired to plot both grid data and non-grid data on the same plot, hidden line removal should be disabled manually after all data are added. Data points with a z value of NaN are internally removed before plotting occurs. ”Holes” in a data set can be constructed by manually setting elements of *z* to this value.

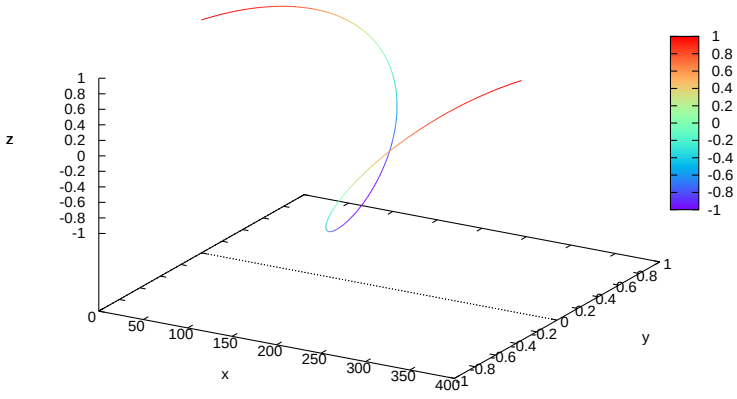
It is important to note that for a 3D grid, the ordering of the z data is important. For calculation of the z values, the *x* value is held constant while *y* is cycled through its range of values. The *x* value is then incremented and *y* is cycled again. This is repeated until all the data are calculated. So, for a 10x20 grid the data shall be ordered as follows:

x1 y1 z1
x1 y2 z2
x1 y3 z3
.
.
.
x1 y18 z18
x1 y19 z19
x1 y20 z20
x2 y1 z21
x2 y2 z22
x2 y3 z23
.
.
.
x10 y18 z198
x10 y19 z199
x10 y20 z200

Example 1

```
/* File: data3D.ch */
#include <math.h>
#include <chplot.h>
#include <numeric.h>

int main() {
    array double x[360], y[360], z[360];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    z = cos(x*M_PI/180);
    plot.data3D(x, y, z);
    plot.plotting();
}
```

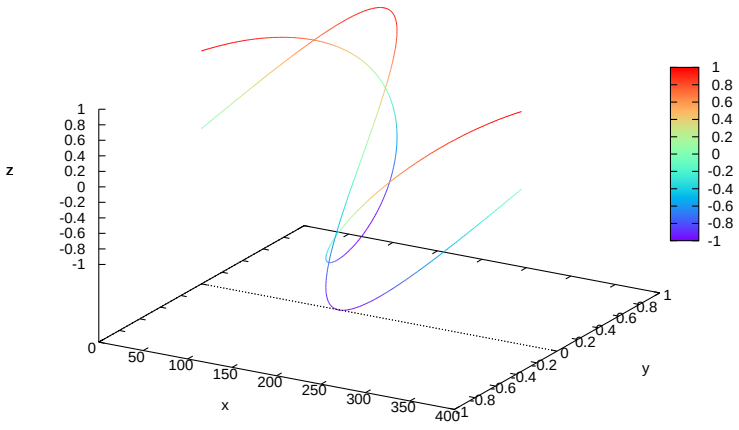
Output



Example 2

```
/* File: data3D_2.ch */
#include <math.h>
#include <chplot.h>
#include <numeric.h>

int main() {
    array double x[360], y[360], z[2][360];
    int i;
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    for(i=0; i<360; i++) {
        z[0][i] = cos(x[i]*M_PI/180);
        z[1][i] = y[i];
    }
    plot.data3D(x, y, z);
    plot.plotType(PLOT_PLOTTYPE_LINES, 1);
    plot.lineType(1, 0, 3); /* set the second data set to
                               use the default line type
                               and a line width three
                               times the default */
    plot.plotting();
}
```

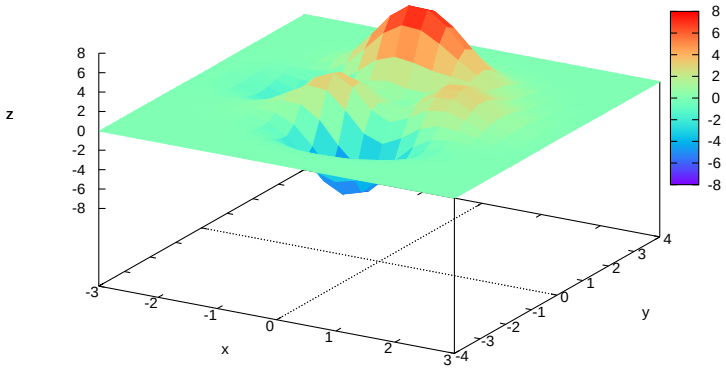
Output



Example 3
Compare with output for examples in **CPlot**::**arrow**(), **CPlot**::**contourLabel**(), **CPlot**::**grid**(), **CPlot**::**removeHiddenLine**(), **CPlot**::**size3D**(), **CPlot**::**changeViewAngle**(), and **CPlot**::**ticsLevel**().

```
/* File: data3D_3.ch */
#include <math.h>
#include <chplot.h>
#include <numeric.h>

int main() {
    double x[20], y[30], z[600];
    int i,j;
    class CPlot plot;

    lindata(-3, 3, x);
    lindata(-4, 4, y);
    for(i=0; i<20; i++) {
        for(j=0; j<30; j++) {
            z[30*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.plotting();

}
```

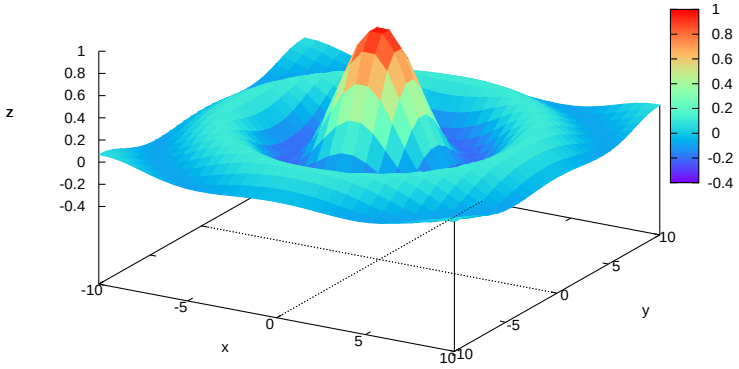
Output



Example 4
Compare with output for example **CPlot**::**axisRange**() and **CPlot**::**colorBox**().

```
/* File: data3D_4.ch */
#include <chplot.h>
#include <math.h>
#include <numeric.h>

int main() {
    double x[30], y[30], z[900];
    double r;
    int i, j;
    class CPlot plot;

    lindata(-10, 10, x);
    lindata(-10, 10, y);
    for(i=0; i<30; i++) {
        for(j=0; j<30; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[30*i+j] = sin(r)/r;
        }
    }
    plot.data3D(x, y, z);
    plot.plotting();
}
```

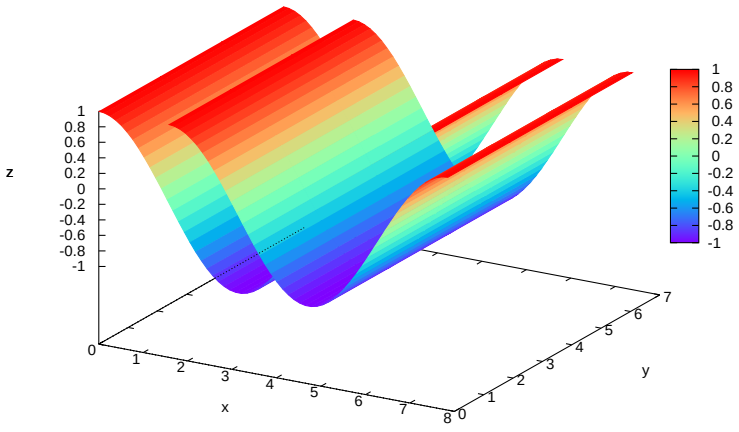
Output



Example 5

```
/* File: data3D_5.ch */
#include <chplot.h>
#include <math.h>
#include <numeric.h>

int main() {
    array double x1[50], x2[50], y[4], z[200];
    int i,j,angle;
    class CPlot plot;

    lindata(0, 2*M_PI, x1);
    lindata(M_PI/2, 2*M_PI+M_PI/2, x2);
    lindata(0, 2*M_PI, y);
    for (i=0;i<50;i++) {
        for (j=0;j<4;j++) {
            z[j+4*i]=cos(x1[i]);          // Z-axis data.
        }
    }
    plot.data3D(x1, y, z);
    plot.data3D(x2, y, z);
    plot.plotting();
}
```

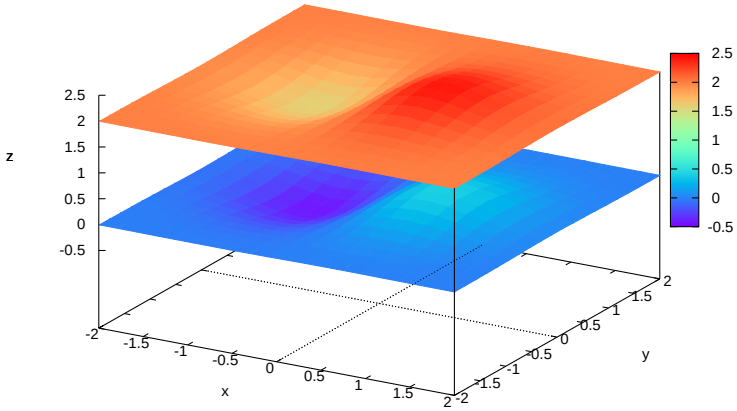
Output



Example 6
Compare with the output for the example **CPlot**::**changeViewAngle**().

```
/* File: data3D_6.ch */
#include <chplot.h>
#include <math.h>
#include <numeric.h>

#define NUMX 20
#define NUMY 20
#define NUMCURVE 2
int main() {
    array double x[NUMX], y[NUMY], z[NUMCURVE][NUMX*NUMY];
    int i, j;
    class CPlot plot;

    lindata(-2, 2, x);
    lindata(-2, 2, y);
    for (i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[0][i*NUMX+j] = x[i]*exp(-x[i]*x[i]-y[j]*y[j]);
            z[1][i*NUMX+j] = z[0][i*NUMX+j] +2;
        }
    }
    plot.data3D(x, y, z);
    plot.plotting();
}
```

Output



See Also
CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**dataFile**(), **CPlot**::**plotting**(), **plotxyz**().

CPlot::data3DCurve

Synopsis
#include <**chplot.h**>
int data3DCurve(**double** *x*[], **double** *y*[], **double** z[], **int** n);

Purpose
Add a set of data for 3D curve to an instance of **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x a one-dimensional array of size n used for the x axis of the plot.

y a one-dimensional array of size n used for the y axis of the plot.

z a one-dimensional array of size n used for the z axis of the plot.

n the number of elements for arrays *x*, *y*, and *z*.

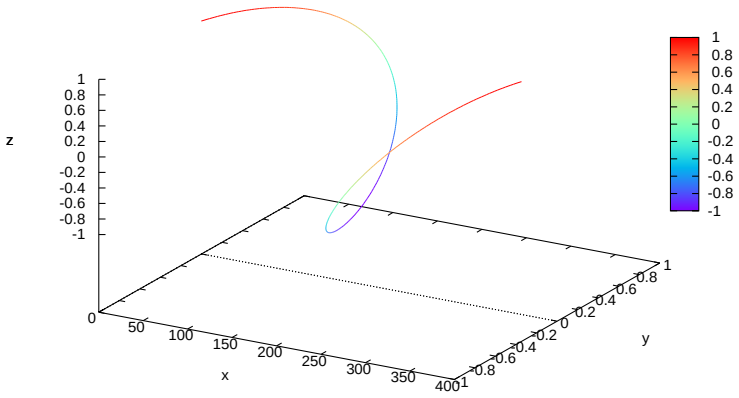
Description
Add a set of data for 3D curve to an instance of **CPlot** class. Arrays x, *y*, and *z* have the same number of elements of size *n*. In a Cartesian coordinate system, these arrays represent data in X-Y-Z coordinates. In a cylindrical coordinate system, *x* represents θ, *y* for z, and *z* for r. In a spherical coordinate system, *x*

represents θ, y for ϕ, and z for r. Data points with a z value of NaN are internally removed before plotting occurs. ”Holes” in a data set can be constructed by manually setting elements of z to this value.

Example 1
Compare with output for examples in **CPlot**::**data3D**().

```
/* File: data3DCurve.ch */
#include <math.h>
#include <chplot.h>
#ifndef M_PI
#define M_PI            3.14159265358979323846
#endif

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM], z[NUM];
    class CPlot plot;

    for(i=0; i< NUM; i++) {
      x[i] = i*10;
      y[i] = sin(x[i]*M_PI/180);
      z[i] = cos(x[i]*M_PI/180);
    }
    plot.data3DCurve(x, y, z, NUM);
    plot.plotting();
    return 0;
}
```

Output

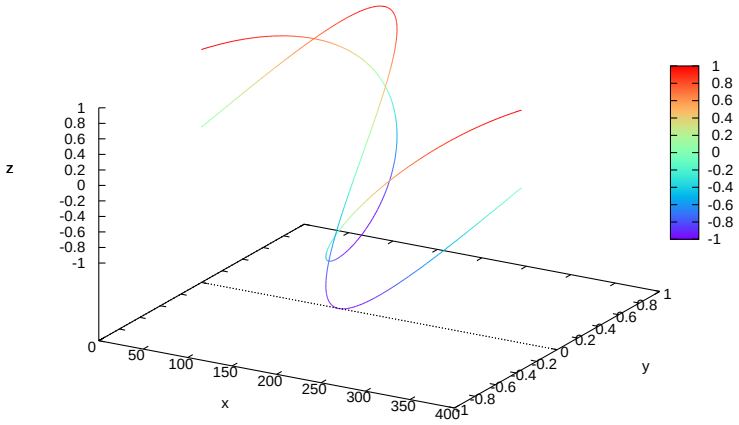


Example 2
Compare with output for examples in **CPlot**::**data3D**().

```
/* File: data3DCurve_2.ch */
#include <math.h>
#include <chplot.h>
#ifndef M_PI
#define M_PI            3.14159265358979323846
```

```
#endif

#define NUM 360
int main() {
    double x[NUM], y[NUM], z1[NUM], z2[NUM];
    int i;
    class CPlot plot;

    for(i=0; i<360; i++) {
        x[i]  = i;
        y[i]  = sin(x[i]*M_PI/180);
        z1[i] = cos(x[i]*M_PI/180);
        z2[i] = y[i];
    }
    plot.data3DCurve(x, y, z1, NUM);
    plot.data3DCurve(x, y, z2, NUM);
    plot.plotType(PLOT_PLOTTYPE_LINES, 1);
    plot.lineType(1, 0, 3); /* set the second data set to
                               use the default line type
                               and a line width three
                               times the default */
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::data2D(), CPlot::data2DCurve(), CPlot::data3D(), CPlot::data3DSurface(), CPlot::dataFile(), CPlot::plotting(), plotxyz().

CPlot::data3DSurface

Synopsis

#include <chplot.h>

int data3DSurface(**double** *x*[], **double** *y*[], **double** *z*[], **int** n, **int** m);

Purpose
Add a set of data for 3D surface to an instance of **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x a one-dimensional array of size n used for the x axis of the plot.

y a one-dimensional array of size m used for the y axis of the plot.

z a one-dimensional array of size nXm

n the number of elements for array *x*.

m the number of elements for array *y*.

Description
Add a set of data for 3D surface plot to an instance of **CPlot** class. If one-dimensional array x has the number of elements of size n, and *y* has size m, *z* shall be a one-dimensional array of size $n_z = n \cdot m$. In a Cartesian coordinate system, arrays *x*, *y*, and *z* represent values in X-Y-Z coordinates, respectively. In a cylindrical coordinate system, arrays *x*, *y*, and *z* represent θ), z, and r coordinates, respectively. In a spherical coordinate system, arrays *x*, *y*, and *z* represent θ), ϕ, and r coordinates, respectively. Hidden line removal is enabled automatically (see **CPlot**::**removeHiddenLine**()). If it is desired to plot both grid data and non-grid data on the same plot, hidden line removal should be disabled manually after all data are added. Data points with a z value of NaN are internally removed before plotting occurs. ”Holes” in a data set can be constructed by manually setting elements of *z* to this value.

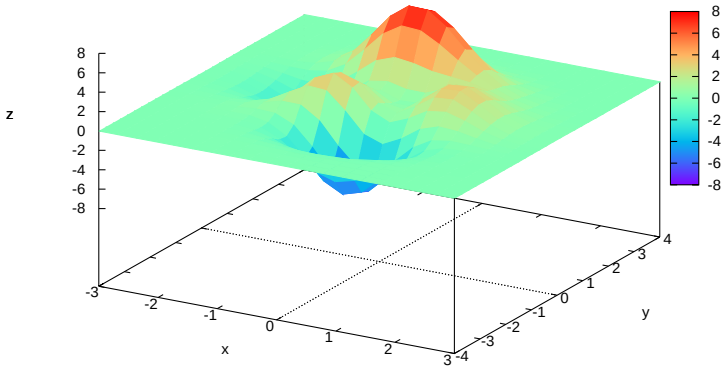
It is important to note that for a 3D grid, the ordering of the z data is important. For calculation of the z values, the *x* value is held constant while *y* is cycled through its range of values. The *x* value is then incremented and *y* is cycled again. This is repeated until all the data are calculated. So, for a 10x20 grid the data shall be ordered as follows:

x1 y1 z1
x1 y2 z2
x1 y3 z3
.
.
.
x1 y18 z18
x1 y19 z19
x1 y20 z20
x2 y1 z21
x2 y2 z22
x2 y3 z23
.
.
.
x10 y18 z198
x10 y19 z199
x10 y20 z200

Example 1
Compare with output for examples in **CPlot**:**data3D**(), **CPlot**::**arrow**(), **CPlot**::**contourLabel**(), **CPlot**::**grid**(), **CPlot**::**removeHiddenLine**(), **CPlot**:**size3D**(), **CPlot**::**changeViewAngle**(), and **CPlot**::**ticsLevel**().

```
/* File: data3DSurface_3.ch */
#include <math.h>
#include <chplot.h>

#define N 20
#define M 30
int main() {
    double x[N], y[M], z[N*M];
    int i,j;
    class CPlot plot;

    for(i=0; i<N; i++) {
      x[i] = -3 + i*6/19.0;  // lindata(-3, 3, x)
    }
    for(i=0; i<M; i++) {
      y[i] = -4 + i*8/29.0;  // lindata(-4, 4, y)
    }
    for(i=0; i<N; i++) {
        for(j=0; j<M; j++) {
            z[M*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, N, M);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**dataFile**(), **CPlot**::**plotting**(), **plotxyz**().

CPlot::dataFile

Synopsis
#include <**chplot.h**>
int dataFile(**string_t** *file*, ... /* [**char** *option*] */);

Syntax
dataFile(*file*)
dataFile(*file*, *option*)

Purpose
Add a data file to an existing instance of the **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameter
file name of the file to be plotted.

option The option for the data file.

Description
Add a data file to an existing plot variable. Each data file corresponds to a single data set. The data file

should be formatted with each data point on a separate line. 2D data are specified by two values per point. An empty line in a 2D data file causes a break of the curve in the plot. Multiple curves can be plotted in this manner, however the plot style will be the same for all curves. 3D data are specified by three values per data point.

For a 3D grid or surface data, each row is separated in the data file by a blank line. By default, hidden lines are not removed for 3D plotting using a data file. Use function **CPlot**::**removeHiddenLine**() to remove hidden lines.

The symbol # will comment out a subsequent text terminated at the end of a line in a data file. For example, a 3 x 3 grid would be represented as follows:

```
# This is a comment line
x1   y1   z1
x1   y2   z2
x1   y3   z3

x2   y1   z4
x2   y2   z5
x2   y3   z6

x3   y1   z7
x3   y2   z8
x3   y3   z9
```

Two empty lines in the data file will cause a break in the plot. Multiple curves or surfaces can be plotted in this manner, however, the plot style will be the same for all curves or surfaces. Member function **CPlot**::**dimension**(3) must be called before 3D data file can be added.

The option for the data file is as follows.

```
using {<entry> {:<entry> {:<entry> ...}}} {'format'}
```

If a format is specified, each datafile record is read using the C library's **scanf**() function, with the specified format string. Otherwise the record is read and broken into columns at spaces or tabs.

The resulting array of data is then sorted into columns according to the entries. Each `<entry>` may be a simple column number, which selects the datum, an expression enclosed in parentheses, or empty. The expression can use `$1` to access the first item read, `$2` for the second item, and so on. A column number of 0 generates a number increasing (from zero) with each point, and is reset upon encountering two blank records. A column number of -1 gives the dataline number, which starts at 0, increments at single blank records, and is reset at double blank records. A column number of -2 gives the index number, which is incremented only when two blank records are found. An empty `<entry>` will default to its order in the list of entries. For example, `using ::4\verb` is interpreted as `using 1:2:4\verb`.

If the `using\verb` list has but a single entry, that `<entry>` will be used for y and the data point number is used for x; for example, `using 1\verb` is identical to `using 0:1\verb`. If the `using\verb` list has two entries, these will be used for x and y. Additional entries are usually plot stype of errors in x and/or y. See **CPlot**::**plotType**() for details about plotting styles that make use of error information.

The C Function **scanf()** accepts several numerical specifications **CPlot** requires all inputs to be double-precision floating-point variables, so `"%lf"` is essentially the only permissible specifier. A format string given by the user must contain at least one such input specifier, and no more than seven of them. scanf() expects to see white space—a blank, tab `("\t")`, newline `("\n")`, or formfeed `("\f")`—between numbers; anything else in the input stream must be explicitly skipped. Note that the use of `"\t", "\n",` or `"\f"` requires use of double-quotes rather than single-quotes.

Examples:

This creates a plot of the sum of the 2nd and 3rd data against the first: The format string specifies comma-rather than space-separated columns.

```
using 1:($2+$3) '%lf,%lf,%lf'
```

In this example the data are read from a using a more complicated format:

```
using "%*lf%lf%*20[^\n]%lf"
```

The meaning of this format is:

```
%*lf        ignore a number
%lf         read a double-precision number (x by default)
%*20[^\n]   ignore 20 non-newline characters
%lf         read a double-precision number (y by default)
```

One trick is to use the C ternary `'?:'\verb` operator to filter data:

```
using 1:($3>10 ? $2 : 1/0)
```

which plots the datum in column two against that in column one provided the datum in column three exceeds ten. `1/0` is undefined; **CPlot** quietly ignores undefined points, so unsuitable points are suppressed.

If timeseries data are being used, the time can span multiple columns. The starting column should be specified. Note that the spaces within the time must be included when calculating starting columns for other data. E.g., if the first element on a line is a time with an embedded space, the y value should be specified as column three.

It should be noted that for three cases a) without option, b) with option of `using 1:2`, c) with option `using ($1):($2)` can be subtly different: 1) if the datafile has some lines with one column and some with two, the first will invent x values when they are missing, the second will quietly ignore the lines with one column, and the third will store an undefined value for lines with one point (so that in a plot with lines, no line joins points across the bad point); 2) if a line contains text at the first column, the first will abort the plot on an error, but the second and third should quietly skip the garbage.

In fact, it is often possible to plot a file with lots of lines of garbage at the top simply by specifying

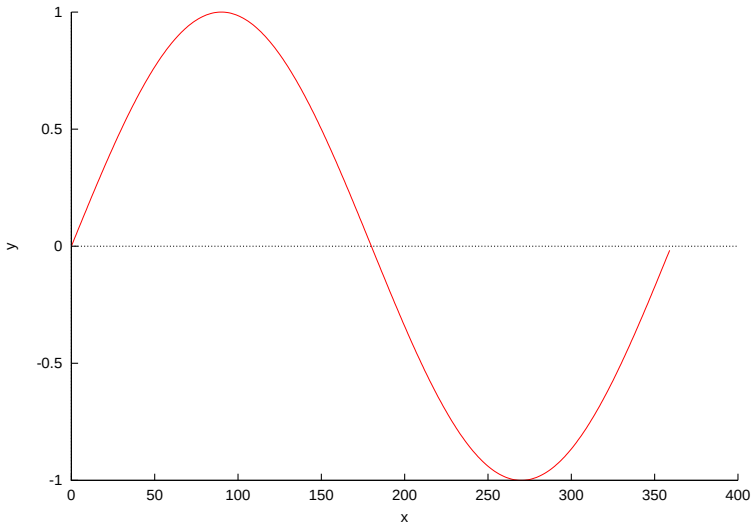
```
        using 1:2
```

However, if you want to leave text in your data files, it is safer to put the comment character '#' in the first column of the text lines.

Example

```
/* File: dataFile.ch */
#include <stdio.h>
#include <chplot.h>
#include <math.h>

int main() {
    string_t file;
    int i;
    class CPlot plot;
    FILE *out;

    file = tmpnam(NULL);                //Create temporary file.
    out=fopen (file,"w");               //Write data to file.
    for(i=0;i<=359;i++) fprintf(out,"%i %f \n",i,sin(i*M_PI/180));
    fclose(out);
    plot.dataFile(file);
    plot.plotting();
    remove(file);
}
```

Output



Examples

See an example on page 149 for **CPlot**:**plotType**(). For comparison with data from **CPlot**::**dataFile**() and **CPlot**::**data**(), see programs on pages 54 and 151 for plot with candlesticks.

See Also

CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**, **CPlot**::**data3DCurve**, **CPlot**::**data3DSurface**,

CPlot::**outputType**(), **CPlot**::**plotting**(), **plotxyf**(), **plotxyzf**().

CPlot::dataSetNum

Synopsis
#include <**chplot.h**>
int dataSetNum();

Purpose
Obtain the current data set number in an instance of **CPlot** class.

Return Value
The current data set number in an instance of **CPlot** class. The first data set number is 0. If there is no data in the instance of the CPlot class, the return value is -1.

Parameters
None.

Description
This function returns the current data set number in an instance of **CPlot** class.

Example

```
/* File: dataSetNum.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int num, numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x, 36);
    y = sin(x*M_PI/180);
    num = plot.dataSetNum();
    printf("The number of data set is %d\n", num);
    plot.data2D(x, y);
    num = plot.dataSetNum();
    printf("The number of data set is %d\n", num);
    plot.data2D(x, 2*y);
    num = plot.dataSetNum();
    printf("The number of data set is %d\n", num);
    plot.plotting();
}
```

Output in console

```
The number of data set is -1
The number of data set is 0
```

```
The number of data set is 1
```

CPlot::deleteData

Synopsis
#include <**chplot.h**>
void deleteData();

Purpose
Delete all plot data of an instance of the **CPlot** class.

Return Value
None.

Parameters
None.

Description
This function frees all memory associated with previously allocated plot data, plot type, legends, plot axes, points, lines, polygons, rectangles, and circles. Unlike **CPlot**::**deletePlots**(), this function does not reset plotting options to their default values. This function allows for the reuse of a single instance of the **CPlot** class to create multiple plots.
See Also
CPlot::**arrow**(), **CPlot**::**circle**(), **CPlot**::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**dataFile**(), **CPlot**::**deletePlots**(), **CPlot**::**ticsLabel**(), **CPlot**::**line**(), **CPlot**::**point**(), **CPlot**::**polygon**, **CPlot**::**rectangle**(), **CPlot**::**text**().

CPlot::deletePlots

Synopsis
#include <**chplot.h**>
void deletePlots();

Purpose
Delete all plot data and reinitialize an instance of the class to default values.

Return Value
None.

Parameters
None.

Description
This function frees all memory associated with previously allocated plot data, plot type, legends, plot axes, text strings, arrows, points, lines, polygons, rectangles, circles, and labeled tic-marks. This function also resets all plotting options to their default values. This function allows for the reuse of a single instance of the **CPlot** class to create multiple plots. This function is used internally by **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyf**(), **plotxyzf**().

See Also
CPlot::**arrow**(), **CPlot**::**circle**(), **CPlot**::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**dataFile**(), **CPlot**::**deleteData**(), **CPlot**::**ticsLabel**(), **CPlot**::**line**(), **CPlot**::**point**(), **CPlot**::**polygon**, **CPlot**::**rectangle**(), **CPlot**::**text**(), **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyf**(), **plotxyzf**().

CPlot::dimension

Synopsis
#include <**chplot.h**>
void dimension(**int** *dim*);

Purpose
Set plot dimension to 2D or 3D.

Return Value
None.

Parameter
dim 2 for 2D and 3 for 3D. Default is 2.

Description
Set the dimension of the plot. The plot dimension should be set before data are added to the plot if member functions **CPlot**::dataThreeD(), **CPlot**::dataThreeDCurve(), or **CPlot**::dataThreeDSurface() are not called before. This member function must be used when 3D plotting data are added by **CPlot**::dataFile() and **CPlot**::**polygon**().

Example
See **CPlot**::**polygon**().

See Also
CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**dataFile**(), **CPlot**::**polygon**().

CPlot::displayTime

Synopsis

#include <**chplot.h**>

void displayTime(**double** *x_offset*, **double** *y_offset*);

Purpose

Display the current time and date.

Return Value

None.

Parameters

x_offset Offset of the time-stamp in the x direction from its default location.

y_offset Offset of the time-stamp in the y direction from its default location.

Description

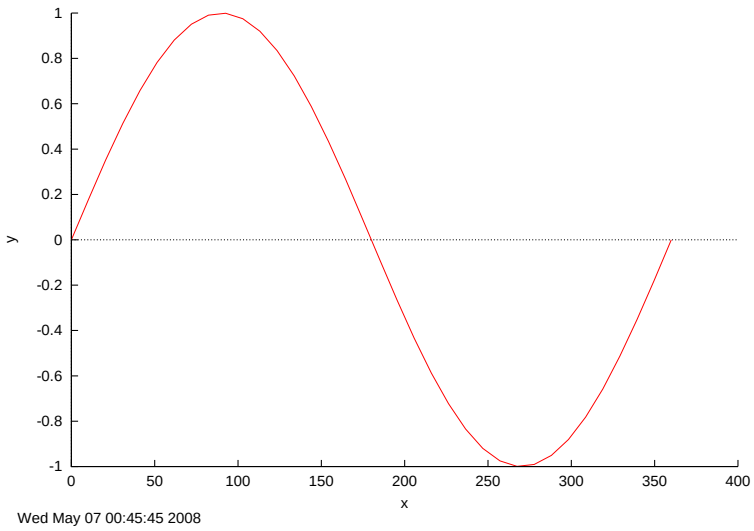
This function places the current time and date near the left margin. The exact location is device dependent. The offset values, *x_offset* and *y_offset*, are in screen coordinates and are measured in characters. For example, if both *x_offset* and *y_offset* are 2, the time will be moved approximately two characters to the right and two characters above the default location.

Example

Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: displayTime.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.displayTime(10,0);
    plot.data2D(x, y);
    plot.plotting();
}
```

Output



CPlot::enhanceText

Synopsis
#include <**chplot.h**>
void enhanceText();

Purpose
Use special symbols in text.

Return Value
None.

Parameters
None.

Description
This function turns on the enhanced text mode for terminal and output files in PostScript, PNG, JPEG, GIF formats that support additional text formatting information embedded in the text string. With this function call, special symbols can be used in text such as those passed arguments of member functions **CPlot**::**label**(), **CPlot**::**legend**(), **CPlot**::**text**(), **CPlot**::**title**().

The syntax for the enhanced text is shown in Figure 2.1. The character codes for the enhanced text is shown in Figure 2.2. Braces can be used to place multiple-character text where a single character is expected, for example, `3^{2x}`. To change the font and/or size, use the full form

```
{/[fontname][=fontsize | *fontscale] text}
```

For example, `{/Symbol=20 p}` is a 20-point π and `{/*0.5 B} is a` B at an half of whatever fontsize is currently in effect. The `'/'` character *must* be the first character after the `'{'` character to use the

enhanced text.

You can access specia l symbols numerically by specifying `\character-code` in octal number, for example, {`/Symbol \160`} is the symbol for π.

You can escape control characters using `\`.

	text	result
Superscripts are denoted by ^:	’10^{-2}’	10^{-2}
Subscripts are denoted by _:	’A_{j,k}’	$A_{j,k}$
Braces are not needed for single characters:	’e^x’	e^{x}
Use @ to align sub- and superscripts:	’x@^2_k’	x_k^2
Put the shorter of the two first:	’x@_0^{-3/2}y’	$x_0^{-3/2}y$
...rather than:	’x@^{-3/2}_0y’	$x_0^{-3/2}y$
Font changes are enclosed in braces:	’{/Helvetica m}’	m
...size, too:	’{/=8 m}’	m
...or both:	’{/Helvetica=18 m}’	m
Characters can be specified by code:	’{\120}’	P
...which is how to get nonkeyboard characters:	’{\267}’	•
Use keyboard characters or codes for other fonts:	’{/Symbol p\271 22/7}’	π≠ 22/7
Everything outside braces is in the default font:	’P = {/Symbol r}kT’	P = ρkT
Space of a given size can be inserted with &:	’<junk>’	<junk>
	’<&{junk}>’	< >
Special characters (^,_,{,},@,&,\) can be escaped by \:	’f\{x,y\}’	f{x,y}
...or \\ if within a double-quoted string:	"f\\{x,y\\}"	f{x,y}

Everything can be done recursively:

the text ’{/Symbol=18 \362@_{/=9.6 0}^{/=12 \245}}
{/Helvetica e^{-{/Symbol m}^2/2} d}{/Symbol m = (p/2)^{1/2}}’

produces the result: $\int_0^{\infty} e^{-\mu^2/2}\, d\mu = (\pi/2)^{1/2}$

Note how font sizes and definitions are preserved across pairs of braces.

図 2.1: Syntax for the enhanced text.

T = text (here Times-Roman) S = Symbol Z = ZapfDingbats E = ISO Latin-1 encoding
(the "E" character set is accessed via a member function CPlot::nativeCmd("set encoding"))

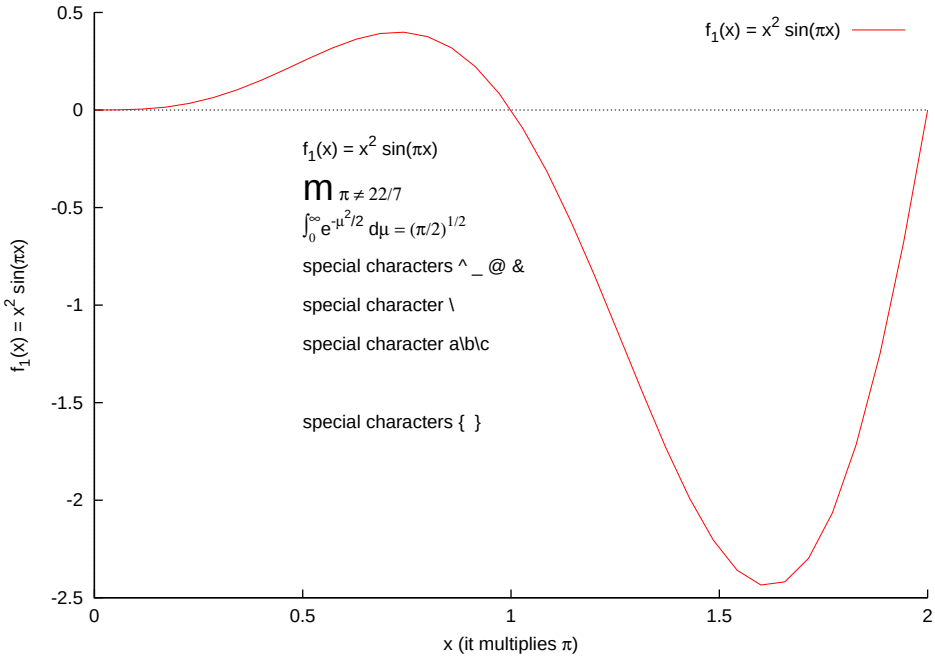
	T	S	Z	E		T	S	Z	E		T	S	Z	E		T	S	Z	E		T	S	Z	E
040					111	I	I	☆	I	162	r	ρ	❒	r	256	fi	→	③	®	327		·	↕	×
041	!	!	✁	!	112	J	ϑ	✪	J	163	s	σ	▲	s	257	fl	↓	④	¯	330		¬	◥	Ø
042	"	∀	✂	"	113	K	K	✯	K	164	t	τ	▼	t	260		°	⑤	°	331		∧	➔	Ù
043	#	#	✃	#	114	L	Λ	✰	L	165	u	υ	◆	u	261	–	±	⑥	±	332		∨	➘	Ú
044	$	∃	✄	$	115	M	M	✱	M	166	v	ϖ	❖	v	262	†	″	⑦	²	333		⇔	→	Û
045	%	%	✆	%	116	N	N	✶	N	167	w	ω	◗	w	263	‡	≥	⑧	³	334		⇐	➔	Ü
046	&	&	✇	&	117	O	O	✷	O	170	x	ξ	❙	x	264	·	×	⑨	´	335		⇑	→	Ý
047	’	∍	✈	’	120	P	Π	✸	P	171	y	ψ	❚	y	265		∝	⑩	µ	336		⇒	➞	Þ
050	(	(	✍	(	121	Q	Θ	✹	Q	172	z	ζ	❯	z	266	¶	∂	❶	¶	337		⇓	➟	ß
051	)	)	✉	)	122	R	P	✺	R	173	{	{	❮	{	267	•	•	❷	·	340		◊	➠	à
052	*	∗	☛	*	123	S	Σ	✻	S	174	|	|	❰	|	270	,	÷	❸	¸	341	Æ	⟨	➡	á
053	+	+	☞	+	124	T	T	✼	T	175	}	}	❝	}	271	„	≠	❹	¹	342		®	➢	â
054	,	,	✌	,	125	U	Y	✽	U	176	~	~	❞	~	272	”	≡	❺	º	343	ª	©	➢	ã
055	-	−	✍	−	126	V	ς	✶	V	220				ı	273	»	≈	❻	»	344		™	➤	ä
056	.	.	✎	.	127	W	Ω	✷	W	221				`	274	…	…	❼	¼	345		∑	➥	å
057	/	/	✏	/	130	X	Ξ	✸	X	222				´	275	‰	|	❽	½	346		⎛	➦	æ
060	0	0	✐	0	131	Y	Ψ	✹	Y	223				ˆ	276		—	❾	¾	347		⎜	➧	ç
061	1	1	✑	1	132	Z	Z	✺	Z	224				˜	277	¿	↵	❿	¿	350	Ł	⎝	➨	è
062	2	2	✒	2	133	[	[	✻	[	225				¯	300		ℵ	①	À	351	Ø	⎡	➩	é
063	3	3	✓	3	134	\	∴	✼	\	226				˘	301	`	ℑ	②	Á	352	Œ	⎢	➪	ê
064	4	4	✔	4	135	]	]	✽	]	227				˙	302	´	ℜ	③	Â	353	º	⎣	➫	ë
065	5	5	✕	5	136	^	⊥	❁	^	230				¨	303	ˆ	℘	④	Ã	354		⎡	➬	ì
066	6	6	✖	6	137	_	_	❂	_	232				°	304	˜	⊗	⑤	Ä	355		⎧	➭	í
067	7	7	✗	7	140	‘		❃	‘	233				¸	305	¯	⊕	⑥	Å	356		⎩	➮	î
070	8	8	✘	8	141	a	α	❄	a	235				˝	306	˘	∅	⑦	Æ	357		⎪	➯	ï
071	9	9	✙	9	142	b	β	✲	b	236				˛	307	˙	∩	⑧	Ç	360				ð
072	:	:	✚	:	143	c	χ	❅	c	237				ˇ	310	¨	∪	⑨	È	361	æ	⟩	➱	ñ
073	;	;	✛	;	144	d	δ	❆	d	240		€			311		⊃	⑩	É	362		⌠	➲	ò
074	<	<	✜	<	145	e	ε	❇	e	241	¡	ϒ	❡	¡	312	°	⊇	❶	Ê	363		⎮	➳	ó
075	=	=	✝	=	146	f	ϕ	❈	f	242	¢	′	❢	¢	313	¸	⊄	❷	Ë	364		⌡	➴	ô
076	>	>	✞	>	147	g	γ	❉	g	243	£	≤	❣	£	314		⊂	❸	Ì	365	ı	⎮	➵	õ
077	?	?	✟	?	150	h	η	❋	h	244	⁄	⁄	❤	¤	315	˝	⊆	❹	Í	366		⎞	➶	ö
100	@	≅	✠	@	151	i	ι	❊	i	245	¥	∞	❥	¥	316	˛	∈	❺	Î	367		⎟	➷	÷
101	A	A	✡	A	152	j	φ	❊	j	246	ƒ	ƒ	❦	¦	317	ˇ	∉	❻	Ï	370	ł	⌡	➸	ø
102	B	B	✢	B	153	k	κ	❋	k	247	§	♣	❧	§	320	—	∠	❼	Đ	371	ø	⎤	➹	ù
103	C	X	⌘	C	154	l	λ	●	l	250	¤	♦	♣	¨	321		∇	❽	Ñ	372	œ	⎥	→	ú
104	D	Δ	⌘	D	155	m	µ	❍	m	251	'	♥	♦	©	322		®	❾	Ò	373	ß	⌡	➺	û
105	E	E	✣	E	156	n	ν	■	n	252	“	♠	♥	ª	323		©	❿	Ó	374		⎦	➻	ü
106	F	Φ	✤	F	157	o	o	❑	o	253	«	↔	♠	«	324		™	➔	Ô	375		⎨	➼	ý
107	G	Γ	✧	G	160	p	π	❒	p	254	‹	←	①	¬	325		∏	→	Õ	376		⎩	➽	þ
110	H	H	★	H	161	q	θ	❑	q	255	›	↑	②	-	326		√	↔	Ö	377				ÿ

図 2.2: character codes for the enhanced text.

Example 1

```
/* File: enhanceText.cpp */
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;
    char funcname[] = "f_1(x) = x^2 sin({/Symbol p}x)";

    for(i=0; i< NUM; i++) {
      x[i] = i*(2.0)/(NUM-1);
      y[i] = x[i]*x[i]*sin(x[i]*M_PI);
    }
    plot.data2DCurve(x, y, NUM);
    plot.enhanceText();
    plot.label(PLOT_AXIS_X, "x (it multiplies {/Symbol p})");
    plot.label(PLOT_AXIS_Y, funcname);
    plot.legend(funcname, 0);
    plot.text(funcname, PLOT_TEXT_LEFT, 0.5, -0.2, 0);
    plot.text("{/Helvetica=28 m} {/Symbol p \271 22/7}", PLOT_TEXT_LEFT, 0.5, -0.4, 0);
    plot.text("{/Symbol=18 \362@_{/=9.6 0}^{/=12 \245}}"
              "{/Helvetica e^{-{/Symbol m}^2/2} d}{/Symbol m = (p/2)^{1/2}}",
               PLOT_TEXT_LEFT, 0.5, -0.6, 0);
    plot.text("special characters \\\\^ \\\\_ \\\\@ \\\\&", PLOT_TEXT_LEFT, 0.5, -0.8, 0);
    plot.text("special character \\134 ", PLOT_TEXT_LEFT, 0.5, -1.0, 0);
    plot.text("special character a\\134b\\\\c", PLOT_TEXT_LEFT, 0.5, -1.2, 0);

    /* for display on the screen */
    plot.text("special characters \\\\{ \\\\} ", PLOT_TEXT_LEFT, 0.5, -1.4, 0);
    /* For postscript file use the format below to create '{' and '}' */
    //plot.text("special characters \\\\173  \\\\175", PLOT_TEXT_LEFT, 0.5, -1.6, 0);
    //plot.outputType(PLOT_OUTPUTTYPE_FILE, "postscript eps color",
                      "../output/outputOption.eps");
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::**label**(), **CPlot**::**legend**(), **CPlot**::**text**(), **CPlot**::).

CPlot::func2D

Synopsis
#include <**chplot.h**>
int func2D(**double** *x0*, **double** *xf*, **int** *n*, **double** (**func*)(**double** *x*));

Purpose
Add a set of data using a function for 2D curve to an instance of **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x0 the initial value for the range of the function.

xf the final value for the range of the function.

n the number of points for the range of the function.

func a pointer to function for adding a set of data.

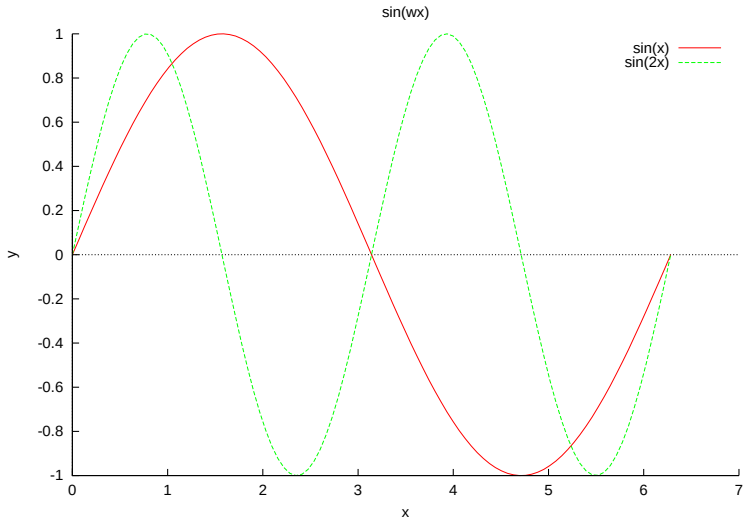
Description
This function adds a set of data using a function `func()` in the range from `x0` to `xf` with n points to a

previously declared instance of the **CPlot** class.

Example 1

```
/* File: func2D.ch */
#include<math.h>
#include<chplot.h>

#define N 100

double omega;
double func(double x) {
   double y;

   y = sin(omega*x);
   return y;
}

int main() {
   double x0, xf;
   CPlot plot;

   x0 = 0;
   xf = 2*M_PI;
   plot.title("sin(wx)");
   plot.func2D(x0, xf, N, sin);
   plot.legend("sin(x)", 0);
   omega = 2;
   plot.func2D(x0, xf, N, func);
   plot.legend("sin(2x)", 1);
   plot.plotting();
}
```

Output



See Also

CPlot::**func3D**(), **CPlot**::**funcp2D**(), **CPlot**::**funcp3D**(), **CPlot**::**data2D**(), **CPlot**::**data3D**(),

CPlot::**data3DCurve**(), **CPlot**::**data3DSurface**(), **fplotxy**().

CPlot::func3D

Synopsis
#include <**chplot.h**>
int func3D(**double** *x0*, **double** *xf*, **double** *y0*, **double** *yf*, **int** *nx*, **int** *ny*, **double** (**func*)(**double** *x*, **double** *y*));

Purpose
Add a set of data using a function for 3D surface to an instance of **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x0 the initial value for the x range of the function.

xf the final value for the x range of the function.

y0 the initial value for the y range of the function.

yf the final value for the y range of the function.

nx the number of points for the x range of the function.

ny the number of points for the y range of the function.

func a pointer to function for adding a set of data.

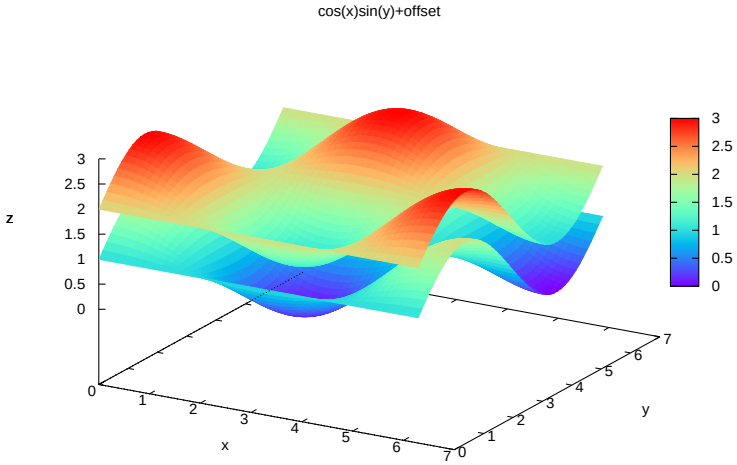
Description
This function adds a set of data using a function `func()` with nx points in the range from `x0` to `xf` for `x` and with ny points in the range from `y0` to `yf` for `y` to a previously declared instance of the **CPlot** class.

Example 1

```
/* File: func3D.ch */
#include<math.h>
#include<chplot.h>

#define NX 50
#define NY 50

double offset;

double func(double x, double y) {
   double z;

   z = cos(x)*sin(y) +offset;
```

```
   return z;
}

int main() {
   double x0, xf, y0, yf;
   CPlot plot;

   x0 = 0;
   xf = 2*M_PI;
   y0 = 0;
   yf = 2*M_PI;
   offset = 1;
   plot.title("cos(x)sin(y)+offset");
   plot.func3D(x0, xf, y0, yf, NX, NY, func);
   offset = 2;
   plot.func3D(x0, xf, y0, yf, NX, NY, func);
   plot.plotting();
}
```

Output



See Also

CPlot::**func2D**(), **CPlot**::**func3D**(), **CPlot**::**funcp2D**(), **CPlot**::**data2D**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **fplotxy**().

CPlot::funcp2D

Synopsis

#include <**chplot.h**>

int funcp2D(**double** *x0*, **double** *xf*, **int** *n*, **double** (***func*)(**double** *x*, **void** * *param*, **void** * *param*);

Purpose

Add a set of data using a function for 2D curve to an instance of **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x0 the initial value for the range of the function.

xf the final value for the range of the function.

n the number of points for the range of the function.

func a pointer to function for adding a set of data.

param a parameter passed to the function. If this argument is not used inside function `func()`, NULL can be passed to this paramenter.

Description
This function adds a set of data using a function with a parameter `func()` in the range from `x0` to `xf` with n points to a previously declared instance of the **CPlot** class.

Example 1

```
/* File: funcp2D.ch */
#include<math.h>
#include<chplot.h>

#define N 100

double func(double x, void *param) {
   double omega, y;

   omega = *(double*)param;
   y = sin(omega*x);
   return y;
}

int main() {
   double x0, xf, omega;
   CPlot plot;

   x0 = 0;
   xf = 2*M_PI;
   omega = 1;
   plot.title("sin(wx)");
   plot.funcp2D(x0, xf, N, func, &omega);
   plot.legend("sin(x)", 0);
   omega = 2;
   plot.funcp2D(x0, xf, N, func, &omega);
   plot.legend("sin(2x)", 1);
   plot.plotting();
}
```

Output
See **CPlot**::**func2D**().

See Also
CPlot::**func2D**(), **CPlot**::**funcp3D**(), **CPlot**::**funcp3D**(), **CPlot**::**data2D**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **fplotxy**().

CPlot::funcp3D

Synopsis
#include <**chplot.h**>
int funcp3D(**double** *x0*, **double** *xf*, **double** *y0*, **double** *yf*, **int** *nx*, **int** *ny*, **double** (**func*)(**double** *x*, **double** *y*, **void** * *param*, **void** * *param*);

Purpose
Add a set of data using a function for 3D surface to an instance of **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x0 the initial value for the x range of the function.

xf the final value for the x range of the function.

y0 the initial value for the y range of the function.

yf the final value for the y range of the function.

nx the number of points for the x range of the function.

ny the number of points for the y range of the function.

func a pointer to function for adding a set of data.

param a parameter passed to the function. If this argument is not used inside function `func()`, NULL can be passed to this paramenter.

Description
This function adds a set of data using a function `func()` with nx points in the range from `x0` to `xf` for `x` and with ny points in the range from `y0` to `yf` for `y` to a previously declared instance of the **CPlot** class. The function has an optional paramter.

Example 1

```
/* File: funcp3D.ch */
#include<math.h>
#include<chplot.h>

#define NX 50
```

```
#define NY 50

double func(double x, double y, void *param) {
   double offset, z;

   offset = *(double*)param;
   z = cos(x)*sin(y) +offset;
   return z;
}

int main() {
   double x0, xf, y0, yf, offset;
   CPlot plot;

   x0 = 0;
   xf = 2*M_PI;
   y0 = 0;
   yf = 2*M_PI;
   offset = 1;
   plot.title("cos(x)sin(y)+offset");
   plot.funcp3D(x0, xf, y0, yf, NX, NY, func, &offset);
   offset = 2;
   plot.funcp3D(x0, xf, y0, yf, NX, NY, func, &offset);
   plot.plotting();
}
```

Output
See **CPlot**::**func3D**().
See Also
CPlot::**func2D**(), **CPlot**::**func3D**(), **CPlot**::**funcp2D**(), **CPlot**::**data2D**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **fplotxy**().

CPlot::getLabel

Synopsis
#include <**chplot.h**>
char * **getLabel**(**int** `axis`);

Purpose
Get the label for a plot axis.

Return Value
The label of the axis.

Parameters
axis The axis with its label to be obtained. Valid values are:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

Description
Get the label of a plot axis.

Example

```
/* File: getLabel.ch */
#include <math.h>
#include <chplot.h>

int main() {
    class CPlot plot;
    string_t xlabel, ylabel, zlabel, title;

    plot.label(PLOT_AXIS_X, "xlabel");
    xlabel = plot.getLabel(PLOT_AXIS_X);
    ylabel = plot.getLabel(PLOT_AXIS_Y);
    zlabel = plot.getLabel(PLOT_AXIS_Z);
    printf("xlabel = %s\n", xlabel);
    printf("ylabel = %s\n", ylabel);
    printf("zlabel = %s\n", zlabel);
    title = plot.getTitle();
    printf("title = %s\n", title);
    plot.title("New Title");
    title = plot.getTitle();
    printf("title = %s\n", title);
}
```

Output

```
xlabel = xlabel
ylabel = y
zlabel = z
title =
title = New Title
```

See Also
CPlot::**label**(), **CPlot**::**title**(), **CPlot**::**getTitle**().

CPlot::getOutputType

Synopsis
#include <chplot.h>
plotinfo_t getOutputType();

Purpose
Get the output type for a plot.

Return Value
The output type in one of the following forms:

PLOT_OUTPUTTYPE_DISPLAY Display the plot on the screen.

PLOT_OUTPUTTYPE_STREAM Output the plot as a standard output stream.

PLOT_OUTPUTTYPE_FILE Output the plot to a file in one of a variety of formats.

Parameters None
Description
Get the output type of a plot.

Example

```
/* File: getOutputType.ch */
#include <math.h>
#include <chplot.h>

int main() {
    class CPlot plot;
    plotinfo_t outputtype;

    outputtype = plot.getOutputType();
    if(outputtype == PLOT_OUTPUTTYPE_DISPLAY)
      printf("output is displayed\n");
    else if(outputtype == PLOT_OUTPUTTYPE_STREAM)
      printf("output is stdout stream\n");
    else if(outputtype == PLOT_OUTPUTTYPE_FILE)
      printf("output is in a file\n");
    return 0;
}
```

Output

```
output is displayed
```

See Also
CPlot::outputType().

CPlot::getSubplot

Synopsis
#include <chplot.h>

class CPlot* getSubplot(**int** *row*, **int** *col*);

Purpose
Get a pointer to an element of a subplot.

Return Value
Returns a pointer to the specified element of the subplot.

Parameters
row The row number of the desired subplot element. Numbering starts with zero.

col The column number of the desired subplot element. Numbering starts with zero.

Description
After the creation of a subplot using **CPlot**::**subplot**(), this function can be used to get a pointer to an element of the subplot. This pointer can be used as a **CPlot** pointer normally would be. Addition of data sets or selection of plotting options are done normally. However, each option only effects the subplot element currently pointed to.

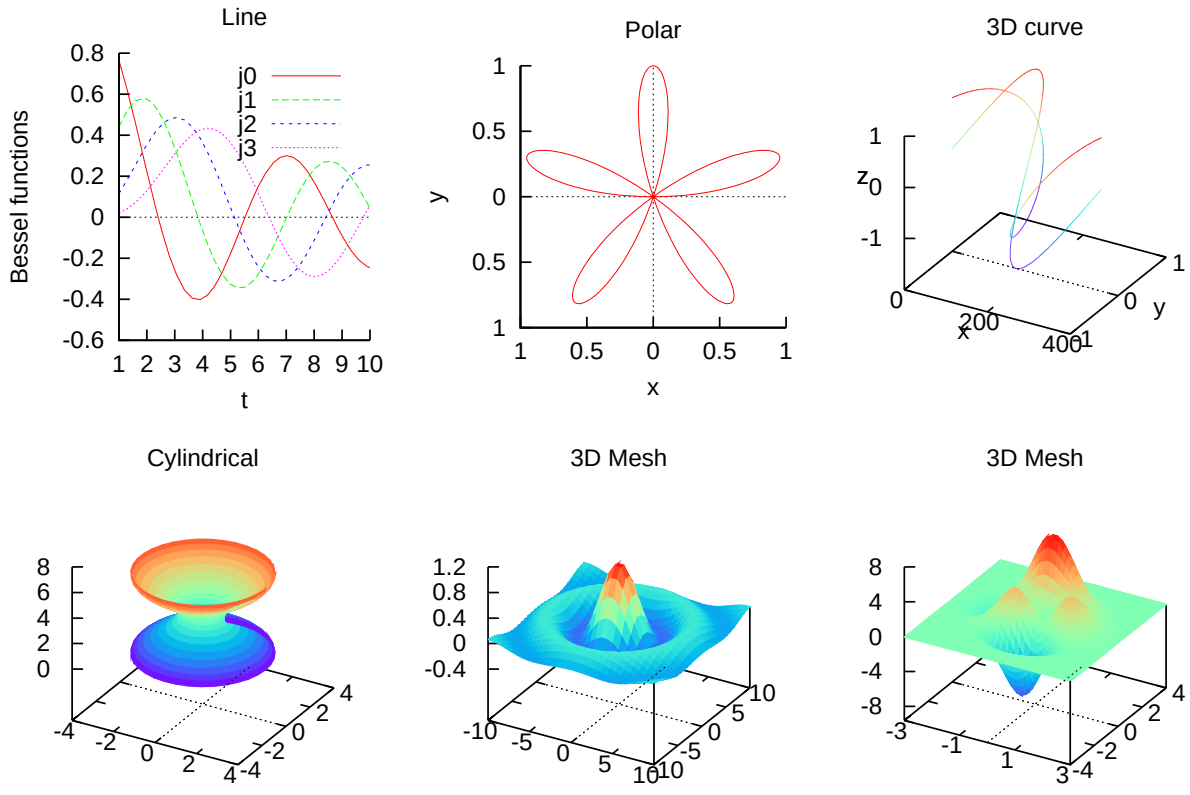
Example

```
/* File: getSubplot.ch */
#include <chplot.h>
#include <math.h>

#define NVAR 4
#define POINTS 50
int main() {
    void derivs(double t, double y[], double dydt[]) {
      dydt[0] = -y[1];
      dydt[1]=y[0]-(1.0/t)*y[1];
      dydt[2]=y[1]-(2.0/t)*y[2];
      dydt[3]=y[2]-(3.0/t)*y[3];
    }

    double t0=1, tf=10, y0[NVAR];
    double t[POINTS], y1[NVAR][POINTS];
    int  points;
    int  datasetnum=0, line_type=4, line_width = 2;
    array double theta2[360], r2[360];
    array double x3[360], y3[360], z3[2][360];
    double theta4[36], z4[20], r4[720];
    double r, x5[30], y5[30], z5[900];
    double x6[40], y6[60], z6[2400];


    int i, j;
    class CPlot subplot, *spl;

    /* plot 1 */
    y0[0]=j0(t0);
    y0[1]=j1(t0);
    y0[2]=jn(2,t0);
```

```
y0[3]=jn(3,t0);
points = odesolve(t, y1, derivs, t0, tf, y0);


/* plot 2 */
lindata(0, M_PI, theta2);
r2 = sin(5*theta2);

/* plot 3 */
lindata(0, 360, x3);
y3 = sin(x3*M_PI/180);
for(i=0; i<360; i++) {
    z3[0][i] = cos(x3[i]*M_PI/180);
    z3[1][i] = y3[i];
}

/* plot 4 */
lindata(0, 360, theta4);
lindata(0, 2*M_PI, z4);
for(i=0; i<36; i++) {
    for(j=0; j<20; j++) {
        r4[i*20+j] = 2+cos(z4[j]);
    }
}

/* plot 5 */
lindata(-10, 10, x5);
lindata(-10, 10, y5);
for(i=0; i<30; i++) {
    for(j=0; j<30; j++) {
        r = sqrt(x5[i]*x5[i]+y5[j]*y5[j]);
        z5[30*i+j] = sin(r)/r;
    }
}

/* plot 6 */
lindata(-3, 3, x6);
lindata(-4, 4, y6);
for(i=0; i<40; i++) {
    for(j=0; j<60; j++) {
        z6[60*i+j] = 3*(1-x6[i])*(1-x6[i])*
                     exp(-(x6[i]*x6[i])-(y6[j]+1)*(y6[j]+1))
        - 10*(x6[i]/5 - x6[i]*x6[i]*x6[i]-pow(y6[j],5))*
                     exp(-x6[i]*x6[i]-y6[j]*y6[j])
        - 1/3*exp(-(x6[i]+1)*(x6[i]+1)-y6[j]*y6[j]);
    }
}

subplot.subplot(2,3); /* create 2 x 3 subplot */
spl = subplot.getSubplot(0,0); /* get subplot (0,0) */
spl->title("Line");
spl->label(PLOT_AXIS_X,"t");
spl->label(PLOT_AXIS_Y,"Bessel functions");
spl->data2D(t, y1);
spl->legend("j0", 0);
spl->legend("j1", 1);
spl->legend("j2", 2);
spl->legend("j3", 3);
```

```
    spl = subplot.getSubplot(0,1); /* get subplot (0,1) */
    spl->title("Polar");
    spl->axisRange(PLOT_AXIS_XY, -1, 1);
    spl->ticsRange(PLOT_AXIS_XY, .5, -1, 1);
    spl->data2D(theta2, r2);
    spl->polarPlot(PLOT_ANGLE_RAD);
    spl->sizeRatio(-1);

    spl = subplot.getSubplot(0,2); /* get subplot (0,2) */
    spl->title("3D curve");
    spl->data3D(x3, y3, z3);
    spl->colorBox(PLOT_OFF);
    spl->axisRange(PLOT_AXIS_X, 0, 400);
    spl->ticsRange(PLOT_AXIS_X, 200, 0, 400);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 1, -1, 1);
    spl->axisRange(PLOT_AXIS_Z, -1, 1);
    spl->ticsRange(PLOT_AXIS_Z, 1, -1, 1);

    spl = subplot.getSubplot(1,0); /* get subplot (1,0) */
    spl->title("Cylindrical");
    spl->data3D(theta4, z4, r4);
    spl->colorBox(PLOT_OFF);
    spl->coordSystem(PLOT_COORD_CYLINDRICAL, PLOT_ANGLE_DEG);
    spl->axisRange(PLOT_AXIS_Z, 0, 8);
    spl->ticsRange(PLOT_AXIS_Z, 2, 0, 8);
    spl->axisRange(PLOT_AXIS_XY, -4, 4);
    spl->ticsRange(PLOT_AXIS_XY, 2, -4, 4);
    spl->label(PLOT_AXIS_XYZ, NULL);

    spl = subplot.getSubplot(1,1); /* get subplot (1,1) */
    spl->title("3D Mesh");
    spl->axisRange(PLOT_AXIS_X, -10, 10);
    spl->ticsRange(PLOT_AXIS_X, 5, -10, 10);
    spl->axisRange(PLOT_AXIS_Y, -10, 10);
    spl->ticsRange(PLOT_AXIS_Y, 5, -10, 10);
    spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
    spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
    spl->data3D(x5, y5, z5);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);

    spl = subplot.getSubplot(1,2); /* get subplot (1,2) */
    spl->title("3D Mesh");
    spl->data3D(x6, y6, z6);
    spl->axisRange(PLOT_AXIS_X, -3, 3);
    spl->ticsRange(PLOT_AXIS_X, 2, -3, 3);
    spl->axisRange(PLOT_AXIS_Y, -4, 4);
    spl->ticsRange(PLOT_AXIS_Y, 2, -4, 4);
    spl->axisRange(PLOT_AXIS_Z, -8, 8);
    spl->ticsRange(PLOT_AXIS_Z, 4, -8, 8);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->ticsLevel(0.1);
    subplot.plotting();
}
```

Output



See Also
CPlot::**subplot**().

CPlot::getTitle

Synopsis
#include <**chplot.h**>
string_t getTitle(void);

Purpose
Get the plot title.

Return Value
The plot title.

Parameters
None.

Description
Get the title of the plot. If no title, `NULL` will be returned.

Example
see **CPlot**::**getLabel**().

See Also
CPlot::**label**(), **CPlot**::**getLabel**(), **CPlot**::**title**().

CPlot::grid

Synopsis
#include <**chplot.h**>
void grid(int *flag*, ... /* [**char** * *option*] */);

Syntax
grid(*flag*)
grid(*flag*, *option*)

Purpose
Enable or disable the display of a grid on the xy plane.

Return Value
None.

Parameters
flag This parameter can be set to:

PLOT_ON Enable the display of the grid.

PLOT_OFF Disable the display of the grid.

option The option for the grid.

Description
Enable or disable the display of a grid on the xy plane. By default, the grid is off. For a polar plot, the polar grid is displayed. Otherwise, the grid is rectangular.

The optional argument `option` of string type with the following values can be used to fine tune the grid based on the argument for set grid command of the gnuplot.

```
{{no}{m}xtics} {{no}{m}ytics} {{no}{m}ztics}
{{no}{m}x2tics} {{no}{m}y2tics}
{{no}{m}cbtics}
{polar {<angle>}}
{layerdefault | front | back}
{ {linestyle <major_linestyle>}
```

```
| {linetype | lt <major_linetype>}
  {linewidth | lw <major_linewidth>}
{ , {linestyle | ls <minor_linestyle>}
    | {linetype | lt <minor_linetype>}
      {linewidth | lw <minor_linewidth>} } }
```

The grid can be enabled and disabled for the major and/or minor tic marks on any axis, and the linetype and linewidth can be specified for major and minor grid lines.

Additionally, a polar grid can be selected for 2-d plots—circles are drawn to intersect the selected tics, and radial lines are drawn at definable intervals. The interval is given in degrees or radians, depending on the argument of **CPlot**::**polarPlot**(). The default polar angle is 30 degrees.

The pertinent tics must be enabled. Otherwise, the plotting engine will quietly ignore instructions to draw grid lines at non-existent tics, but they will appear if the tics are subsequently enabled.

The ‘linetype‘ is followed by an integer index representing the line type for drawing. The line type varies depending on the terminal type used (see **CPlot**::**outputType**). Typically, changing the line type will change the color of the line or make it dashed or dotted. All terminals support at least six different line types. The ‘linewidth‘ is followed by a scaling factor for the line width. The line width is ‘linewidth‘ multiplied by the default width. Typically the default width is one pixel.

If no linetype is specified for the minor gridlines, the same linetype as the major gridlines is used.

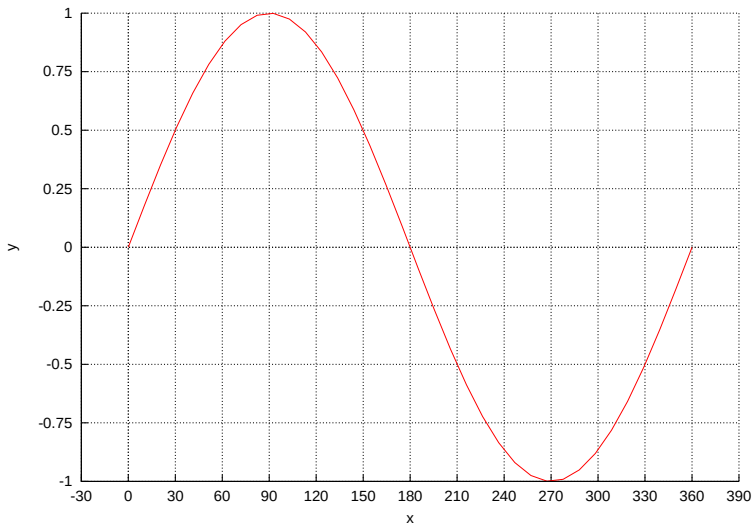
If `"front"` is given, the grid is drawn on top of the graphed data. If `"back"` is given, the grid is drawn underneath the graphed data. Using `"front"` will prevent the grid from being obscured by dense data. The default setup, `"layerdefault"`, is equivalent to `"back"` for 2d plots. In 3D plots the default is to split up the grid and the graph box into two layers: one behind, the other in front of the plotted data and functions.

For 3D plot, Z grid lines are drawn on the bottom of the plot.

Example 1
Compare with the output for the example in **CPlot**::**axisRange**().

```
/* File: grid.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    int i;
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);                        // Y-axis data.
    plot.axisRange(PLOT_AXIS_X, -30, 390);
    plot.ticsRange(PLOT_AXIS_X, 30, -30, 390);
    plot.axisRange(PLOT_AXIS_Y, -1, 1);
    plot.ticsRange(PLOT_AXIS_Y, .25, -1, 1);
    plot.grid(PLOT_ON);
    plot.data2D(x, y);
    plot.plotting();
}
```

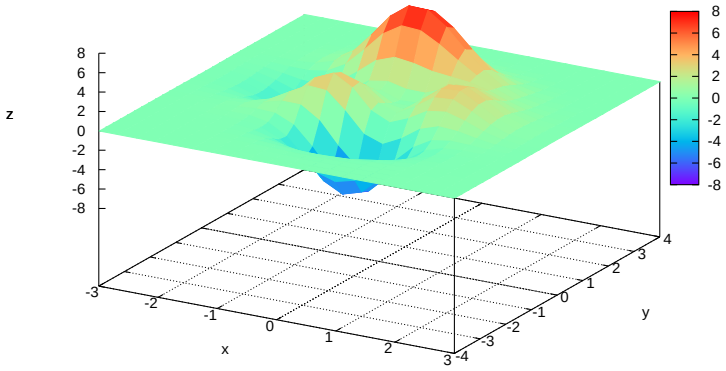
Output



Example 2
Compare with the output for examples in **CPlot**::**data3D**() and **CPlot**::**data3DSurface**().

```
/* File: grid_2.ch */
#include <math.h>
#include <chplot.h>

int main() {
    double x[20], y[30], z[600];
    int i,j;
    class CPlot plot;

    lindata(-3, 3, x);
    lindata(-4, 4, y);
    for(i=0; i<20; i++) {
        for(j=0; j<30; j++) {
            z[30*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.grid(PLOT_ON);
    plot.plotting();

}
```

Output



Example 3
An example with grids for both y and y2 axes on page 23 for **CPlot**:**axes**().
Example 4
An example with grids on the front for a filled curve on page 157 for **CPlot**:**plotType**().

CPlot::isUsed

Synopsis
#include <**chplot.h**>
int isUsed();

Purpose
Test if an instance of the **CPlot** class has been used.

Return Value
Returns `true` if the **CPlot** instance has been previously used, returns `false` otherwise.

Parameters
None.

Description
This function determines if an instance of the **CPlot** class has previously been used. The function actually tests if data have previously been added to the instance, by checking if an internal pointer in the CPlot class is NULL. If the pointer is not NULL, then the instance has been used. This function is used internally by **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyf**(), and **plotxyzf**().

See Also
fplotxy(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyf**(), and **plotxyzf**().

CPlot::label

Synopsis
#include <**chplot.h**>
void label(**int** `axis`, **string_t** *label*);

Purpose
Set the labels for the plot axes.

Return Value
None.

Parameters
axis The axis to be set. Valid values are:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

label label of the axis.

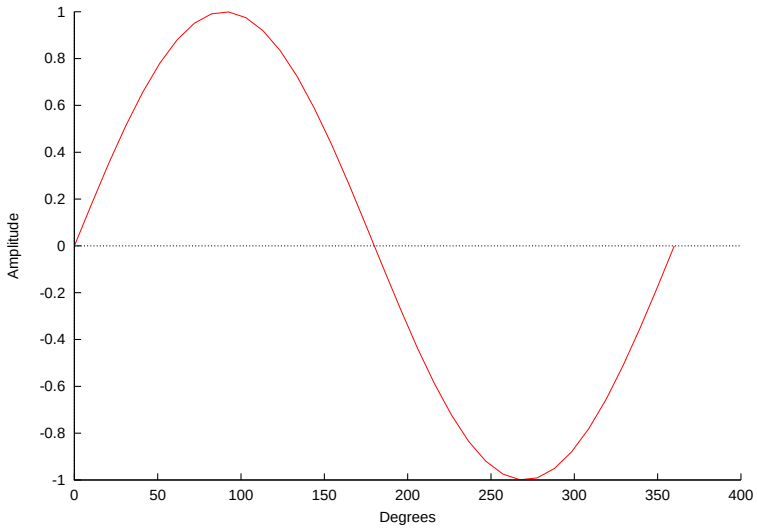
Description
Set the plot axis labels. The label for the z-axis can be set only for a 3D plot. If no label is desired, the value of `NULL` can be used as an argument. For example, function call `plot.label(PLOT_AXIS_XY, NULL)` will suppress the labels for both x and y axes of the plot. By default, labels are “x”, “y”, and “z” for the first x, y, and z axes, respectively.

Example
Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: label.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    string_t xlabel="Degrees",
             ylabel="Amplitude";
    class CPlot plot;
```

```
    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.label(PLOT_AXIS_X, xlabel);
    plot.label(PLOT_AXIS_Y, ylabel);
    plot.data2D(x, y);
    plot.plotting();
}
```

Output



See Also
CPlot::**getLabel**(), **CPlot**::**title**().
CPlot::**getTitle**().

CPlot::legend

Synopsis
#include <**chplot.h**>
void legend(**string_t** *legend*, **int** *num*);

Purpose
Specify a legend string for a previously added data set.

Return Value
None.

Parameters
legend The legend string.

num The data set the legend is added to.

Description

The legend string is added to a plot legend located in the upper-right corner of the plot by default. Numbering of the data sets starts with zero. New legends will replace previously specified legends.

This member function shall be called after plotting data have been added by member functions **CPlot**::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**dataFile**().

Bugs

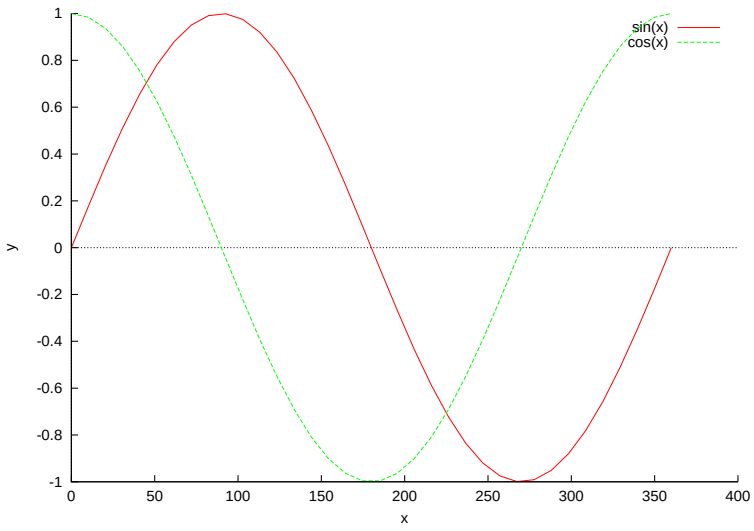
The legend string may not be displayed for a data set with a single point. To suppress the legend string completely, passs value of `" "` to the argument `legend`. Use **CPlot**::**text**() to add a legend for the data set.

Example

Compare with the output for the example in **CPlot**::**legendLocation**().

```
/* File: legend.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints], y2[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    y2= cos(x*M_PI/180);
    plot.data2D(x, y);
    plot.data2D(x, y2);
    plot.legend("sin(x)", 0);
    plot.legend("cos(x)", 1);
    plot.plotting();
}
```

Output



See Also

CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(),

CPlot::**data3DSurface**(), **CPlot**::**dataFile**(), **CPlot**::**legendLocation**(), **CPlot**::**legendOption**().

CPlot::legendLocation

Synopsis
#include <**chplot.h**>
void legendLocation(**double** *x*, **double** *y*, ... /* [**double** *z*] */);

Syntax
legendLocation(*x*, *y*)
legendLocation(*x*, *y*, *z*)

Purpose
Specify the plot legend (if any) location

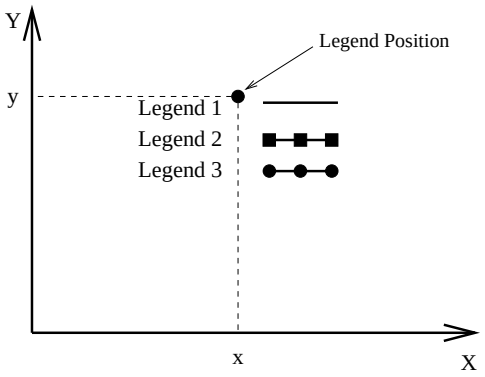
Return Value
None.

Parameters
x The x coordinate of the legend.

y The y coordinate of the legend.

z The z coordinate of the legend.

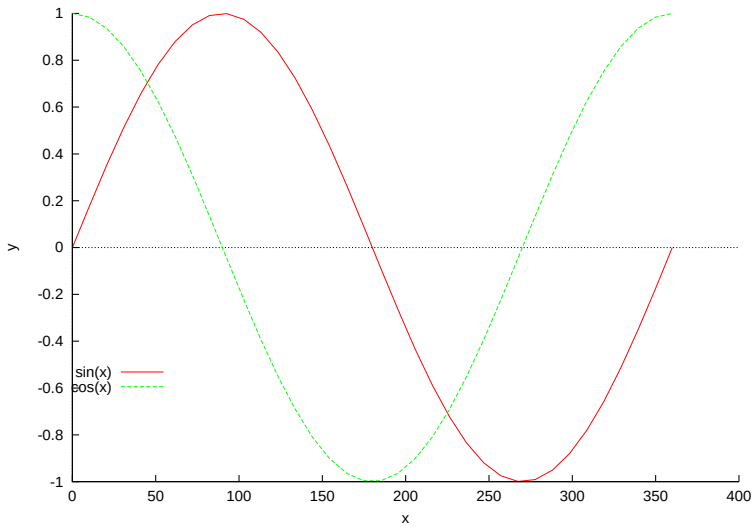
Description
This function specifies the position of the plot legend using plot coordinates. The position specified is the location of the top right of the box for the markers and labels of the legend, as shown below. By default, the location of the legend is near the upper-right corner of the plot. For a two-dimensional plot, the third argument is not necessary.



Example
Compare with the output for the example in **CPlot**::**legend**().

```
/* File: legendLocation.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints], y2[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    y2= cos(x*M_PI/180);
    plot.data2D(x, y);
    plot.data2D(x, y2);
    plot.legend("sin(x)", 0);
    plot.legend("cos(x)", 1);
    plot.legendLocation(60, -.5);
    plot.plotting();
}
```

Output



See Also
CPlot::**legend**(), **CPlot**::**legendOption**().

CPlot::legendOption

Synopsis
#include <**chplot.h**>
void legendOption(**char** * *option*);

Syntax
legendOption(*option*)

Purpose
Set options for legends of a plot.

Return Value
None.

Parameters
option The options for legends of a plot.

Description
The optional argument `option` of string type with the following values can be used to fine tune the legends of a plot.

```
{{inside | outside} | {lmargin | rmargin | tmargin | bmargin}
 | {at <position>}}
{left | right | center} {top | bottom | center}
{vertical | horizontal} {Left | Right}
{{no}reverse} {{no}invert}
{samplen <sample_length>} {spacing <vertical_spacing>}
{width <width_increment>}
{height <height_increment>}
{{no}autotitle {columnheader}}
{{no}box { {linestyle | ls <line_style>}
           | {linetype | lt <line_type>}
             {linewidth | lw <line_width>}}}
```

Legends are stacked according to ‘vertical‘ or ‘horizontal‘. In the case of ‘vertical‘, the legendskey occupy as few columns as possible. That is, legends are aligned in a column until running out of vertical space at which point a new column is started. In the case of ‘horizontal‘, the legends occupy as few rows as possible.

By default the legends are placed in the upper right inside corner of the graph. The keywords ‘left‘, ‘right‘, ‘top‘, ‘bottom‘, ‘center‘, ‘inside‘, ‘outside‘, ‘lmargin‘, ‘rmargin‘, ‘tmargin‘, ‘bmargin‘(, ‘above‘, ‘over‘, ‘below‘ and ‘under‘) may be used to automatically place the legends in other positions of the graph. Also an ‘at `<positionx>`‘ may be given to indicate precisely where the plot should be placed. In this case, the keywords ‘left‘, ‘right‘, ‘top‘, ‘bottom‘ and ‘center‘ serve an analogous purpose for alignment.

To understand positioning, the best concept is to think of a region, i.e., inside/outside, or one of the margins. Along with the region, keywords ‘left/center/right‘ (l/c/r) and ‘top/center/bottom‘ (t/c/b) control where within the particular region the legends should be placed.

When in ‘inside‘ mode, the keywords ‘left‘ (l), ‘right‘ (r), ‘top‘ (t), ‘bottom‘ (b), and ‘center‘ (c) push the legends out toward the plot boundary as illustrated:

```
t/l   t/c   t/r

c/l    c    c/r

b/l   b/c   b/r
```

When in ‘outside‘ mode, automatic placement is similar to the above illustration, but with respect to the view, rather than the graph boundary. That is, a border is moved inward to make room for the key outside of the plotting area, although this may interfere with other labels and may cause an error on some devices. The particular plot border that is moved depends upon the position described above and the stacking direction. For options centered in one of the dimensions, there is no ambiguity about which border to move. For the corners, when the stack direction is ‘vertical‘, the left or right border is moved inward appropriately. When the stack direction is ‘horizontal‘, the top or bottom border is moved inward appropriately.

The margin syntax allows automatic placement of legends regardless of stack direction. When one of the margins ‘lmargin‘ (lm), ‘rmargin‘ (rm), ‘tmargin‘ (tm), and ‘bmargin‘ (bm) is combined with a single, non-conflicting direction keyword, the following illustrated positions may contain the legends:

```
        l/tm   c/tm   r/tm

t/lm                    t/rm

c/lm                    c/rm

b/lm                    b/rm

        l/bm   c/bm   r/bm
```

Keywords ‘above‘ and ‘over‘ are synonymous with ‘tmargin‘. For version compatibility, ‘above‘ or ‘over‘ without an additional l/c/r or stack direction keyword uses ‘center‘ and ‘horizontal‘. Keywords ‘below‘ and ‘under‘ are synonymous with ‘bmargin‘. For compatibility, ‘below‘ or ‘under‘ without an additional l/c/r or stack direction keyword uses ‘center‘ and ‘horizontal‘. A further compatibility issue is that ‘outside‘ appearing without an additional t/b/c or stack direction keyword uses ‘top‘, ‘right‘ and ‘vertical‘ (i.e., the same as t/rm above).

The `<position>` can be a simple x,y,z as in previous versions, but these can be preceded by one of five keywords (‘first‘, ‘second‘, ‘graph‘, ‘screen‘, ‘character‘) which selects the coordinate system in which the position of the first sample line is specified. See ‘coordinates‘ for more details. The effect of ‘left‘, ‘right‘, ‘top‘, ‘bottom‘, and ‘center‘ when `<position>` is given is to align the legends as though it were text positioned using the label command, i.e., ‘left‘ means left align with key to the right of ¡position¿, etc.

Justification of the labels within the key is controlled by ‘Left‘ or ‘Right‘ (default is ‘Right‘). The text and sample can be reversed (‘reverse‘) and a box can be drawn around the legend (`‘box {...}‘`) in a specified ‘linetype‘ and ‘linewidth‘. Note that not all terminal drivers support linewidth selection, though.

By default the first plot label is at the top of the legends and successive labels are entered below it. The ‘invert‘ option causes the first label to be placed at the bottom of the legends, with successive labels entered above it. This option is useful to force the vertical ordering of labels in the legends to match the order of box types in a stacked histogram.

The length of the sample line can be controlled by ‘samplen‘. The sample length is computed as the sum of the tic length and `<sample_length>` times the character width. ‘samplen‘ also affects the positions of point samples in the legends since these are drawn at the midpoint of the sample line, even if the sample line itself is not drawn.

The vertical spacing between lines is controlled by ‘spacing‘. The spacing is set equal to the product of the pointsize, the vertical tic size, and `<vertical_spacing>`. The program will guarantee that the vertical spacing is no smaller than the character height.

The `<width_increment>` is a number of character widths to be added to or subtracted from the length of the string. This is useful only when you are putting a box around the legends and you are using control characters in the text. CPlot class simply counts the number of characters in the string when computing the box width; this allows you to correct it.

The `<height_increment>` is a number of character heights to be added to or subtracted from the height of the key box. This is useful mainly when you are putting a box around the legends, otherwise it can be used to adjust the vertical shift of automatically chosen legend position by `<height_increment>/2`.

The defaults for legends are ‘on‘, ‘right‘, ‘top‘, ‘vertical‘, ‘Right‘, ‘noreverse‘, ‘noinvert‘, ‘samplen 4‘, ‘spacing 1.25‘, and ‘nobox‘. The default `<linetype>` is the same as that used for the plot borders.

The legends are drawn as a sequence of lines, with one plot described on each line. On the right-hand side (or the left-hand side, if ‘reverse‘ is selected) of each line is a representation that attempts to mimic the way the curve is plotted. On the other side of each line is the text description (the line title), obtained from the member function **CPlot**::legend(). The lines are vertically arranged so that an imaginary straight line divides the left- and right-hand sides of the key. It is the coordinates of the top of this line that are specified in the argument of `option` or member function **CPlot**::**legendLocation**(). For a 2D plot, only the x and y coordinates are used to specify the line position. For a 3D plot, x, y and z are all used as a 3-d location mapped using the same mapping as the graph itself to form the required 2-d screen position of the imaginary line.

When using the TeX or PostScript drivers, or similar drivers where formatting information is embedded in the string, CPlot class is unable to calculate correctly the width of the string for the legend positioning. If the legends are to be positioned at the left, it may be convenient to use the combination ‘left Left reverse‘. The box and gap in the grid will be the width of the literal string.

For a 3D countour, the contour labels will be listed in the legends.

Examples:

This places the legends at coordinates 2,3.5,2 in the default (first) coordinate system:

```
plot.legendOption("2,3.5,2");
```

This places the legends below the graph:

```
plot.legendOption("below");
```

This places the legends in the bottom left corner, left-justifies the text, and draws a box around it in linetype 3:

```
plot.legendOption("left bottom Left box 3");
```

Examples

See three examples on pages 153, 154, and 155 for **CPlot**:**plotType**().

See Also

CPlot::**legend**(), **CPlot**::**legendLocation**().

CPlot::line

Synopsis
#include <chplot.h>
int line(double *x1*, **double** *y1*, **double** *z1*, **double** *x2*, **double** *y2*, **double** *z2*);

Purpose
Add a line to a plot.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x1 The x coordinate of the first endpoint of the line.

y1 The y coordinate of the first endpoint of the line.

z1 The z coordinate of the first endpoint of the line. This parameter is ignored for 2D plots.

x2 The x coordinate of the second endpoint of the line.

y2 The y coordinate of the second endpoint of the line.

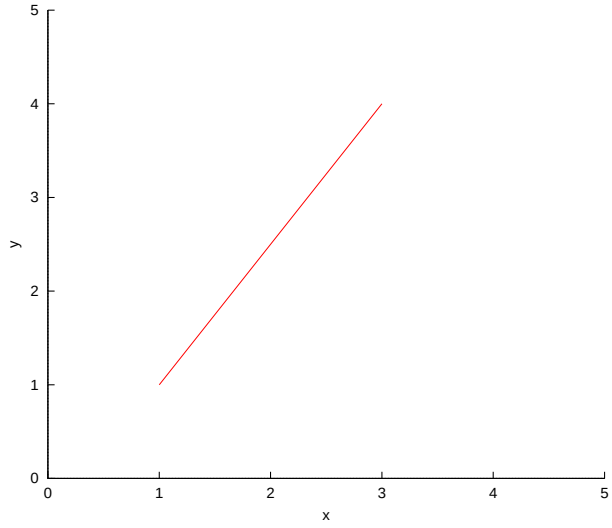
z2 The z coordinate of the second endpoint of the line. This parameter is ignored for 2D plots.

Description
This function adds a line to a plot. It is a convenience function for creation of geometric primitives. A line added with this function is counted as a data set for later calls to **CPlot**::**legend**() and **CPlot**::**plotType**(). For 2D rectangular and 3D cartesian plots, (*x1*, *y1*, *z1*) and (*x2*, *y2*, *z2*) are the coordinates of the endpoints of the line, specified in units of the x, y, and z axes. However, for 2D plots, *z1* and *z2* are ignored. For 2D polar and 3D cylindrical plots, the endpoints are specified in polar coordinates where *x* is θ, *y* is r, and *z* is unchanged. Again, for 2D plots, *z1* and *z2* are ignored. For 3D plots with spherical coordinates *x* is θ, *y* is ϕ and *z* is r.

Example 1

```
/* File: line.ch */
#include <chplot.h>

int main(){
    double x1 = 1, y1 = 1, x2 = 3, y2 = 4;
    class CPlot plot;

    plot.line(x1, y1, 0, x2, y2, 0);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_XY, 0, 5);
    plot.plotting();
}
```

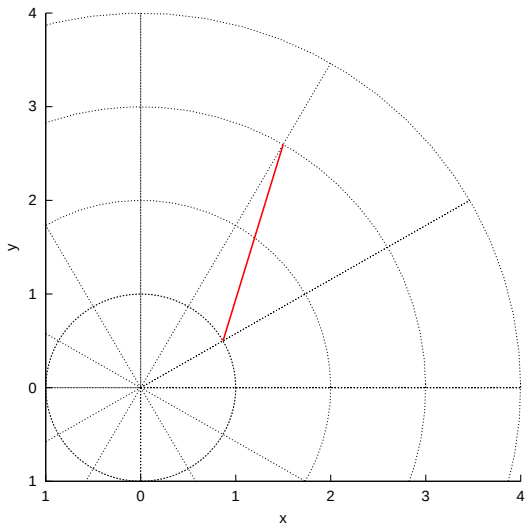
Output



Example 2

```
/* File: line_2.ch */
#include <chplot.h>
#include <math.h>

int main(){
    double theta1 = 30, theta2 = 60, r1 = 1, r2 = 3;
    class CPlot plot;

    plot.grid(PLOT_ON);
    plot.polarPlot(PLOT_ANGLE_DEG);
    plot.line(theta1, r1, 0, theta2, r2, 0);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.lineType(0, 1, 3);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_XY, -1, 4);
    plot.plotting();
}
```

Output



See Also
CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**circle**(), **CPlot**::**outputType**(), **CPlot**::**plotType**(), **CPlot**::**point**(), **CPlot**::**polygon**(), **CPlot**::**rectangle**().

CPlot::lineType

Synopsis
#include <**chplot.h**>
void lineType(**int** *num*, **int** *line_type*, **int** *line_width*], ... /* [**char** *line_color*] */);

Syntax
lineType(*num*, *line_type*, *line_width*)
lineType(*num*, *line_type*, *line_width*, *line_color*)

Purpose
Set the line type, width, and color for lines, impulses, steps, etc.

Return Value
None.

Parameters

num The data set to which the line type, width, and color apply.

line_type An integer index representing the line type for drawing. Use the same value for different curves so that each curve with the same style, and same color by default.

line_width A scaling factor for the line width. The line width is *line_width* multiplied by the default width.

line_color color for the line.

Description

Set the desired line type, width, and color for a previously added data set.

Numbering of the data sets starts with zero. The line style and/or marker type for the plot are selected automatically. The default line type, width, and color can be changed by this member function.

The *line_type* specifies an index for the line type used for drawing the line. The line type varies depending on the terminal type used (see **CPlot**::**outputType**). Typically, changing the line type will change the color of the line or make it dashed or dotted. All terminals support at least six different line types. By default, the line type is 1. The *line_width* specifies the line width. The line width is *line_width* multiplied by the default width. Typically the default width is one pixel.

An optional fourth argument can specify the color of a line by a color name or RGB value, such as `"blue"` or `"#0000ff"` for color blue. The default line type, width, and color can be changed by the function call

```
plot.lineType(num, linetype, linewidth, "blue");
```

The color of the line is specified as blue in this example. The valid color names and their corresponding GRB values are listed below.

```
 Color Name         Hexadecimal R   G    B
                                values
-----------------------------------------
 white               #ffffff = 255 255 255
 black               #000000 =   0   0   0
 gray0               #000000 =   0   0   0
 grey0               #000000 =   0   0   0
 gray10              #1a1a1a =  26  26  26
 grey10              #1a1a1a =  26  26  26
 gray20              #333333 =  51  51  51
 grey20              #333333 =  51  51  51
 gray30              #4d4d4d =  77  77  77
 grey30              #4d4d4d =  77  77  77
 gray40              #666666 = 102 102 102
 grey40              #666666 = 102 102 102
 gray50              #7f7f7f = 127 127 127
 grey50              #7f7f7f = 127 127 127
 gray60              #999999 = 153 153 153
 grey60              #999999 = 153 153 153
 gray70              #b3b3b3 = 179 179 179
 grey70              #b3b3b3 = 179 179 179
 gray80              #cccccc = 204 204 204
 grey80              #cccccc = 204 204 204
 gray90              #e5e5e5 = 229 229 229
```

```
grey90              #e5e5e5 = 229 229 229
gray100             #ffffff = 255 255 255
grey100             #ffffff = 255 255 255
gray                #bebebe = 190 190 190
grey                #bebebe = 190 190 190
light-gray          #d3d3d3 = 211 211 211
light-grey          #d3d3d3 = 211 211 211
dark-gray           #a9a9a9 = 169 169 169
dark-grey           #a9a9a9 = 169 169 169
red                 #ff0000 = 255   0   0
light-red           #f03232 = 240  50  50
dark-red            #8b0000 = 139   0   0
yellow              #ffff00 = 255 255   0
light-yellow        #ffffe0 = 255 255 224
dark-yellow         #c8c800 = 200 200   0
green               #00ff00 =   0 255   0
light-green         #90ee90 = 144 238 144
dark-green          #006400 =   0 100   0
spring-green        #00ff7f =   0 255 127
forest-green        #228b22 =  34 139  34
sea-green           #2e8b57 =  46 139  87
blue                #0000ff =   0   0 255
light-blue          #add8e6 = 173 216 230
dark-blue           #00008b =   0   0 139
midnight-blue       #191970 =  25  25 112
navy                #000080 =   0   0 128
medium-blue         #0000cd =   0   0 205
royalblue           #4169e1 =  65 105 225
skyblue             #87ceeb = 135 206 235
cyan                #00ffff =   0 255 255
light-cyan          #e0ffff = 224 255 255
dark-cyan           #008b8b =   0 139 139
magenta             #ff00ff = 255   0 255
light-magenta       #f055f0 = 240  85 240
dark-magenta        #8b008b = 139   0 139
turquoise           #40e0d0 =  64 224 208
light-turquoise     #afeeee = 175 238 238
dark-turquoise      #00ced1 =   0 206 209
pink                #ffc0cb = 255 192 203
light-pink          #ffb6c1 = 255 182 193
dark-pink           #ff1493 = 255  20 147
coral               #ff7f50 = 255 127  80
```

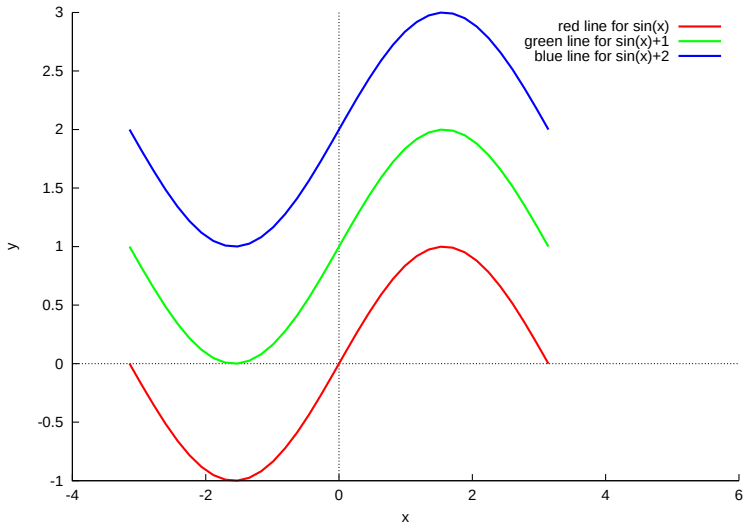
```
  light-coral         #f08080 = 240 128 128
  orange-red          #ff4500 = 255  69   0
  salmon              #fa8072 = 250 128 114
  light-salmon        #ffa07a = 255 160 122
  dark-salmon         #e9967a = 233 150 122
  aquamarine          #7fffd4 = 127 255 212
  khaki               #f0e68c = 240 230 140
  dark-khaki          #bdb76b = 189 183 107
  goldenrod           #daa520 = 218 165  32
  light-goldenrod     #eedd82 = 238 221 130
  dark-goldenrod      #b8860b = 184 134  11
  gold                #ffd700 = 255 215   0
  beige               #f5f5dc = 245 245 220
  brown               #a52a2a = 165  42  42
  orange              #ffa500 = 255 165   0
  dark-orange         #ff8c00 = 255 140   0
  violet              #ee82ee = 238 130 238
  dark-violet         #9400d3 = 148   0 211
  plum                #dda0dd = 221 160 221
  purple              #a020f0 = 160  32 240
```

Example 1

```
/* File: lineType.cpp */
#include <math.h>
#include <chplot.h>

int main() {
    array double x[36], y[36];
    int line_type = 1, line_width = 4, datasetnum = 0;
    CPlot plot;

    lindata(-M_PI, M_PI, x);
    y = sin(x);
    plot.data2D(x, y);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width, "red");
    plot.legend("red line for sin(x)", datasetnum);

    y = sin(x)+1;
    plot.data2D(x, y);
    plot.plotType(PLOT_PLOTTYPE_LINES, ++datasetnum);
    plot.lineType(datasetnum, line_type, line_width, "green");
    plot.legend("green line for sin(x)+1", datasetnum);

    y = sin(x)+2;
    plot.data2D(x, y);
    plot.plotType(PLOT_PLOTTYPE_LINES, ++datasetnum);
    plot.lineType(datasetnum, line_type, line_width, "blue");
    plot.legend("blue line for sin(x)+2", datasetnum);

    plot.axisRange(PLOT_AXIS_X, -4, 6);
```

```
    plot.plotting();
}
```

Output



See Also
CPlot::**plotType**(), **CPlot**::**pointType**().

CPlot::margins

Synopsis
#include <**chplot.h**>
void margins(**double** *left*, **double** *right*, **double** *top*, **double** *bottom*);

Purpose
Set the size of the margins around the edge of the plot.

Return Value
None.

Parameters
left The size of the left margin in character width.

right The size of the right margin in character width.

top The size of the top margin in character height.

bottom The size of the bottom margin in character height.

Description

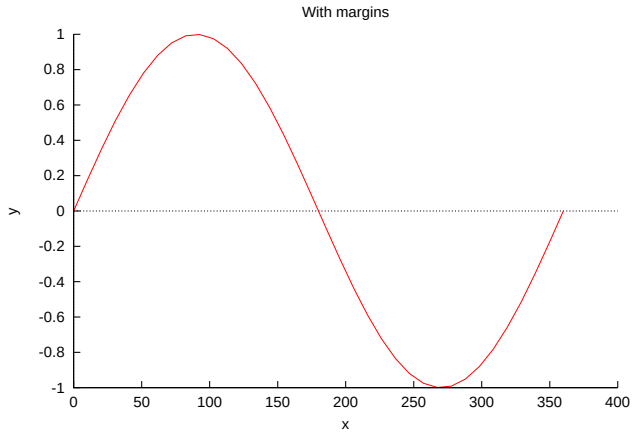
By default, the plot margins are calculated automatically. They can be set manually with this function. Specifying a negative value for a margin causes the default value to be used.

Example

Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: margins.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.data2D(x, y);
    plot.margins(15, 10, 5, 7);
    plot.title("With margins");
    plot.plotting();
}
```

Output



See Also

CPlot::**borderOffsets**().

CPlot::origin

Synopsis

#include <**chplot.h**>

void origin(**double** *x_orig*, **double** *y_orig*);

Purpose
Set the location of the origin for the drawing of the plot.

Return Value
None.

Parameters
x_orig The x coordinate of the origin for the drawing of the plot.

y_orig The y coordinate of the origin for the drawing of the plot.

Description
This function specifies the location of the origin for the drawing of the plot. This is the location of the bottom left corner of the bounding box of the plot, not the location of the origin of the plot coordinate system. The values *x_org* and *y_org* are specified as numbers between 0 and 1, where (0,0) is the bottom left of the plot area (the plot window) and (1,1) is the top right. This function is used internally by method **CPlot**::**subplot**() in conjunction with method **CPlot**::**size**().

See Also
CPlot::**getSubplot**(), **CPlot**::**size**(), **CPlot**::**subplot**().

CPlot::outputType

Synopsis
#include <**chplot.h**>
void outputType(**int** *outputtype*, ... /* [**string_t** *terminal*, **string_t** *filename*] */);

Syntax
outputType(*outputtype*)
outputType(**PLOT_OUTPUTTYPE_DISPLAY**)
outputType(**PLOT_OUTPUTTYPE_STREAM**, *terminal*)
outputType(**PLOT_OUTPUTTYPE_FILE**, *terminal*, *filename*)

Purpose
Set the output type for a plot.

Return Value
None.

Parameters
outputtype This can have any of the following values:

PLOT_OUTPUTTYPE_DISPLAY Display the plot on the screen. The plot is displayed in its own separate window. A plot window can be closed by pressing the 'q' key in the X-Windows system.

PLOT_OUTPUTTYPE_STREAM Output the plot as a standard output stream. This output type is useful for CGI (Common Gateway Interface) when a Ch program is used as CGI script in a Web server.

PLOT_OUTPUTTYPE_FILE Output the plot to a file in one of a variety of formats. If this output option is selected two additional arguments are necessary: the *terminal* type and *filename*.

terminal Supported terminal types when gnuplot is used as a plotting engine are as follow:

Terminal	Description
aifm	Adobe Illustrator 3.0.
corel	EPS format for CorelDRAW.
dxf	AutoCAD DXF.
dxy800a	Roland DXY800A plotter.
eepic	Extended LaTeXpicture.
emtex	LaTeXpicture with emTeX specials.
epson-180dpi	Epson LQ-style 24-pin printer with 180dpi.
epson-60dpi	Epson LQ-style 24-pin printers with 60dpi.
epson-lx800	Epson LX-800, Star NL-10 and NX-100.
excl	Talaris printers.
fig	Xfig 3.1.
gif	GIF file format.
gpic	gpic/groff package.
hp2648	Hewlett Packard HP2647 an HP2648.
hp500c	Hewlett Packard DeskJet 500c.
hpdj	Hewlett Packard DeskJet 500.
hpgl	HPGL output.
hpljii	HP LaserJet II.
hppj	HP PaintJet and HP3630 printers.
latex	LaTeXpicture.
mf	MetaFont.
mif	Frame Maker MIF 3.00.
nec-cp6	NEC CP6 and Epson LQ-800.
okidata	9-pin OKIDATA 320/321 printers.
pcl5	Hewlett Packard LaserJet III.
pbm	Portable BitMap.
png	Portable Network Graphics.

postscript	Postscript.
pslatex	LaTeXpicture with postscript specials.
pstricks	LaTeXpicture with PSTricks macros.
starc	Star Color Printer.
tandy-60dpi	Tandy DMP-130 series printers.
texdraw	LaTeXtexdraw format.
tgif	TGIF X-Window drawing format.
tpic	LaTeXpicture with tpic specials.

`aifm` Output an Adobe Illustrator 3.0 file. The format for the *terminal* string is:

```
"aifm [colormode] [\"fontname\"] [fontsize]"
```

`colormode` can be `color` or `monochrome`.
`fontname` is the name of a valid PostScript font.
`fontsize` is the size of the font in points.

Defaults are `monochrome`, `"Helvetica"` and 14pt.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "aifm color", "plot.aif");`

`corel` Output EPS format for CorelDRAW.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "corel", "plot.eps");`

`dxf` Output AutoCAD DXF file.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "dxf", "plot.dxf");`

`dxy800a` Output file for Roland DXY800A plotter.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "dxy800a", "plot.dxy");`

`eepic` Output extended LaTeXpicture.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "eepic", "plot.tex");`

`emtex` Output LaTeXpicture with emTeX specials.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "emtex", "plot.tex");`

`epson-180dpi` Epson LQ-style 24-pin printers with 180dpi.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "epson-180dpi", "plot");`

`epson-60dpi` Epson LQ-style 24-pin printers with 60dpi.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "epson-60dpi", "plot");`

`epson-lx800` Epson LX-800, Star NL-10 and NX-100.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "epson-lx800", "plot");`

`excl` Talaris printers such as EXCL Laser printer and 1590.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "excl", "plot");`

`fig` Xfig 3.1 file. The format for the *terminal* string is:

```
"fig [colormode] [size] [pointsmax num_points] [orientation]
[units] [fontsize fntsize] [size xsize ysize] [thickness width]
[depth layer]"
```

`colormode` can be `monochrome` or `color`
`size` can be `small` or `big`
`num_points` is the maximum number of points per polyline.
`orientation` can be `landscape` or `portrait`. `units` can be `metric` or `inches`.
`fontsize` is the size of the text font in points. Must be preceded by the `fontsize` keyword.
`xsize` and `ysize` set the size of the drawing area. Must be preceded by the `size` keyword.
`width` is the line thickness. Must be preceded by the `thickness` keyword.
`layer` is the line depth. Must be preceded by the `depth` keyword.

Default values are: `monochrome small pointsmax 1000 landscape inches fontsize 10 thickness 1 depth 10`.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "fig color big", "plot.fig");`

`gif` GIF file format. The format for the *terminal* string is:

```
"gif [transparent] [interlace] [font_size] [size x,y]
[color0 color1 color2 ...]
```

Specifying the `transparent` keyword will generate a transparent GIF. By default, white is the transparent color.
Specifying the `interlace` key word will generate an interlaced GIF.
`font_size` is `small` (6x12 pixels), `medium` (7x13 pixels), or `large` (8x16 pixels).
`x` and `y` are the image size in pixels. Must be preceded by the `size` keyword.
`colors` are specified in the format ”xrrggbb” where x is the character ”x” and ”rrggbb” are the RGB components of the color in hexadecimal. A maximum of 256 colors can be set. If the GIF is transparent, the first color is used as the transparent color.

The default values are: `small size 640,480`

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "gif size 1024,768", "plot.gif");`

`gpic` Output for use with the Free Software Foundation gpic/groff p package. The format for the *terminal* string is:

```
"gpic [x] [y]"
```

where `x` and `y` are the location of the origin. Default is (0,0).

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "gpic 5 5", "plot.gpic");`

`hp2633a` Hewlett Packard HP2623A.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hp2633a", "plot");`

`hp2648` Hewlett Packard HP2647 an HP2648.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hp2648", "plot");`

`hp500c` Hewlett Packard DeskJet 500c. The format for the *terminal* string is:

```
"hp500c [resolution] [compression]"
```

`resolution` can be `75`, `100`, `150`, or `300` dpi.
`compression` can be `rle` or `tiff`.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hp500c 100 tiff", "plot");`

`"hpdj`: Hewlett Packard DeskJet 500. The format for the *terminal* string is:

`"hp500c [resolution]"`

`resolution` can be `75`, `100`, `150`, or `300` dpi. Default is `75`.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hpdj 100", "plot");`

`hpgl` Produces HPGL output for devices such as the HP7475A plotter. The format for the *terminal* string is:

`"hpgl [num_of_pens] [eject]"`

`num_of_pens` is the number of available pens. The default is 6.
`eject` is a keyword that tells the plotter to eject a page when done.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hpgl 4", "plot");`

`hpljii` HP LaserJet II

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hpljii", "plot");`

item[`hppj`] HP PaintJet and HP3630 printers. The format for the *terminal* string is:

`"hppj [font]"`

`font` can be `FNT5X9`, `FNT9x17`, or `FNT13X25`. Default is `FNT9x17`.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hppj FNT5X9", "plot");`

`latex` Output a LaTeXpicture. The format of the *terminal* string is:

`"latex [font] [size]"`

where `font` can be `courier` or `roman` and `size` can be any point size. Default is `roman` 10pt.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "latex", "plot.tex");`

`mf` Output file for the MetaFont program.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "mf", "plot.mf");`

`mif` Output Frame Maker MIF file format version 3.00. The format of the *terminal* string is:

`"mif pentype curvetype"`

`pentype` can be: `colour` or `monochrome`.

`curvetype` can be:

`polyline` curves drawn as continuous lines

`vectors` curves drawn as a collection of vectors

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "mif colour vectors", "plot.mif");`

`nec-cp6` Generic 24-pin printer such as NEC CP6 and Epson LQ-800.

The format for the *terminal* string is:

`"nec-cp6a [option]`

`option` can be `monochrome`, `colour`, or `draft`.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "nec-cp6a draft", "plot");`

`okidata` 9-pin OKIDATA 320/321 printers.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "okidata", "plot");`

`pcl5` Hewlett Packard LaserJet III. This actually produces HPGL-2. The format of the *terminal* string is:

`"pcl5 [mode] [font] [fontsize]"`

`mode` is `landscape` or `portrait`.
`font` is `stick`, `univers`, or `cg_times`.

`fontsize` is the font size in points.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "pcl5 landscape", "plot");`

`pbm` Output a Portable BitMap. The format for the *terminal* string is:

```
"pbm [fontsize] [colormode]"
```

`fontsize` is `small`, `medium`, or `large`.
`colormode` is `monochrome`, `gray`, or `color`.

Example:

```
plot.outputType(PLOT_OUTPUTTYPE_FILE, "pbm medium gray",
"plot.pbm");
```

`png` Portable Network Graphics format. The format of the *terminal* string is:

```
"png [fontsize]"
```

`fontsize` can be `small`, `medium`, or `large`.

Default is `small` and `monochrome` with the output size of 640x480 pixel. Use member function **CPlot**::**size**() to change the size of the plot.

Example:

```
plot.outputType(PLOT_OUTPUTTYPE_FILE, "png", "plot.png");
```

`postscript` This produces a postscript file. The format for the *terminal* string is:

```
"postscript [mode] [colormode] [dash] [\"fontname\"] [fontsize]"
```

`mode` can be `landscape`, `portrait`, `eps`, or `default`
`colormode` can be `color` or `monochrome`.
`dash` can be `solid` or `dashed`.
`fontname` is the name of a valid PostScript font.
`fontsize` is the size of the font in points.

The `default` mode is `landscape`, `monochrome`, `dashed`, `"Helvetica"`, 14pt.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE,"postscript eps \"Times\" 11", "plot.eps");`

`pslatex` Output LaTeXpicture with postscript specials.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "pslatex", "plot.tex");`

`pstricks` Output LaTeXpicture with PSTricks macros.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "pstricks", "plot.tex");`

`starc` Star Color Printer.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "starc", "plot");`

`tandy-60dpi` Tandy DMP-130 series printers.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "tandy-60dpi", "plot");`

`texdraw` Output LaTeXtexdraw format.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "texdraw", "plot.tex");`

`tgif` Output TGIF X-Window drawing format.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "tgif", "plot.tgif");`

`tpic` Output LaTeXpicture with tpic specials.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "tpic", "plot.tex");`

filename The filename the plot is saved to. On machines that support `popen()` functions, the output can also be piped to another program by placing the '|' character in front of the command name and using it as the *filename*. For example, on Unix systems, setting *terminal* to "`postscript`" and *filename* to "`|lp`" could be used to send a plot directly to a postscript printer.

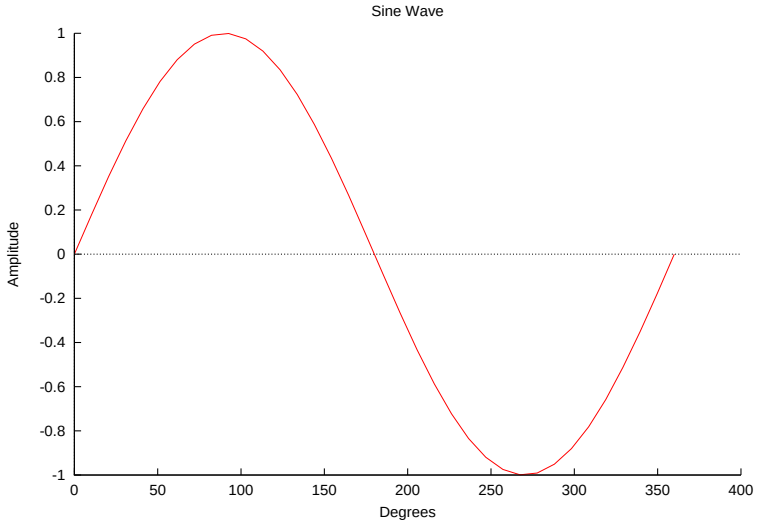
Description

This function is used to display a plot on the screen, save a plot to a file, or generate a plot to the stdout stream in GIF format for use on the World Wide Web. By default, the output type is **PLOT_OUTPUTTYPE_DISPLAY**.

Example 1

```
/* File: outputType.ch */
/* a plot is created in postscript file plot.eps */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    string_t title="Sine Wave",                // Define labels.
             xlabel="Degrees",
             ylabel="Amplitude";
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);                       // Y-axis data.
    plot.title(title);
    plot.label(PLOT_AXIS_X, xlabel);
    plot.label(PLOT_AXIS_Y, ylabel);
    plot.data2D(x, y);
    plot.outputType(PLOT_OUTPUTTYPE_FILE, "postscript eps color", "plot.eps");
    plot.plotting();
}
```

Output

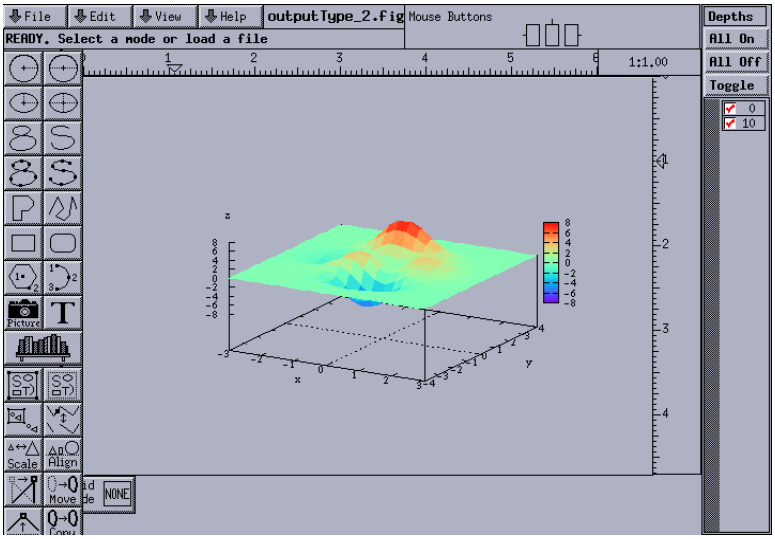


Example 2

```
/* File: outputType_2.ch */
/* In this example, the plot is saved as an xfig file first.
   Next the xfig program is invoked. The plot can then be edited using xfig in Unix */
#include <math.h>
#include <chplot.h>

int main() {
    double x[20], y[30], z[600];
    int i,j;
    class CPlot plot;

    lindata(-3, 3, x);
    lindata(-4, 4, y);
```

```
    for(i=0; i<20; i++) {
        for(j=0; j<30; j++) {
            z[30*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.outputType(PLOT_OUTPUTTYPE_FILE, "fig color", "../output/outputType_2.fig");
    plot.plotting();

    xfig ../output/outputType_2.fig&

}
```

Output



Example 3
To run this code as a CGI program in a Web server, place outputType.ch in your `cgi-bin` directory. If you place outputType.ch in a different directory, change `/cgi-bin` to specify the correct location. In your Web browser, open the html file.

```
#!/bin/ch

/* This flle is called outputType.ch, a plot is created in png format  */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    string_t title="Sine Wave",
             xlabel="Degrees",
             ylabel="Amplitude";
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    setbuf(stdout, NULL);
    printf("Content-type: image/png\n\n");
```

```
    plot.title(title);
    plot.label(PLOT_AXIS_X, xlabel);
    plot.label(PLOT_AXIS_Y, ylabel);
    plot.data2D(x, y);
    plot.outputType(PLOT_OUTPUTTYPE_STREAM, "png color");
    plot.plotting();
}
```

outputType.html

```
<! this file is called outputType.html >
<html>
<head>
<title>
Example Plot
</title>
</head>

<img src="/cgi-bin/outputType.ch">

</html>
```

CPlot::plotType

Synopsis
#include <**chplot.h**>
void plotType(**int** *plot_type*, **int** *num*, ... /* [**char** *option*]] */);

Obsolete Synopsis
#include <**chplot.h**>
void plotType(**int** *plot_type*, **int** *num*, ... /* [[**int** *line_type*, **int** *line_width*],
[**int** *point_type*, **int** *point_size*], [**char** *option*]] */);

Syntax
plotType(**PLOT_PLOTTYPE_LINES**, *num*)
plotType(**PLOT_PLOTTYPE_IMPULSES**, *num*)
plotType(**PLOT_PLOTTYPE_FSTEPS**, *num*)
plotType(**PLOT_PLOTTYPE_HISTEPS**, *num*)
plotType(**PLOT_PLOTTYPE_POINTS**, *num*)
plotType(**PLOT_PLOTTYPE_LINESPOINTS**, *num*)
plotType([**PLOT_PLOTTYPE_STEPS**, *num*)
plotType(**PLOT_PLOTTYPE_DOTS**, *num*)
plotType(**PLOT_PLOTTYPE_SURFACES**, *num*)
plotType(**PLOT_PLOTTYPE_FINANCEBARS**, *num*)
plotType(**PLOT_PLOTTYPE_BOXES**, *num*)
plotType(**PLOT_PLOTTYPE_BOXERRORBARS**, *num*)
plotType(**PLOT_PLOTTYPE_BOXXYERRORBARS**, *num*)
plotType(**PLOT_PLOTTYPE_XERRORBARS**, *num*)

plotType(**PLOT_PLOTTYPE_XYERRORBARS**, *num*)
plotType(**PLOT_PLOTTYPE_YERRORBARS**, *num*)
plotType(**PLOT_PLOTTYPE_XERRORLINES**, *num*)
plotType(**PLOT_PLOTTYPE_XYERRORLINES**, *num*)
plotType(**PLOT_PLOTTYPE_YERRORLINES**, *num*)
plotType(**PLOT_PLOTTYPE_VECTORS**, *num*)
plotType(**PLOT_PLOTTYPE_VECTORS**, *num*, *option*)
plotType(**PLOT_PLOTTYPE_CANDLESTICKS**, *num*)
plotType(**PLOT_PLOTTYPE_CANDLESTICKS**, *num*, *option*)
plotType(**PLOT_PLOTTYPE_FILLEDCURVES**, *num*)
plotType(**PLOT_PLOTTYPE_FILLEDCURVES**, *num*, *option*)

Obsolte Syntax
plotType(**PLOT_PLOTTYPE_LINES**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_IMPULSES**, *num*, *line_type*, *line_width*)
plotType([**PLOT_PLOTTYPE_STEPS**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_FSTEPS**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_HISTEPS**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_POINTS**, *num*, *point_type*, *point_size*)
plotType(**PLOT_PLOTTYPE_LINESPOINTS**, *num*, *line_type*, *line_width*, *point_type*, *point_size*)
plotType(**PLOT_PLOTTYPE_DOTS**, *num*, *point_type*)
plotType(**PLOT_PLOTTYPE_FINANCEBARS**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_BOXES**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_BOXERRORBARS**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_BOXXYERRORBARS**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_XERRORBARS**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_XYERRORBARS**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_YERRORBARS**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_XERRORLINES**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_XYERRORLINES**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_YERRORLINES**, *num*, *line_type*, *line_width*)

Use
lineType(*num*, *line_type*, *line_width*)
lineType(*num*, *line_type*, *line_width*, *line_color*)
pointType(*num*, *pint_type*, *point_size*)
pointType(*num*, *point_type*, *point_size*, *point_color*)

Purpose
Set the plot type for a data set.

Return Value

None.

Parameters

plot_type The plot type. Valid values are:

PLOT_PLOTTYPE_LINES Data points are connected with a line. This is the default for 2D plots. When this plot type used for 3D plot, the surface is meshed with wire frames.

PLOT_PLOTTYPE_IMPULSES Display vertical lines from the x-axis (for 2D plots) or the xy plane (for 3D plots) to the data points.

PLOT_PLOTTYPE_STEPS Adjacent points are connected with two line segments, one from (x1,y1) to (x2,y1), and a second from (x2,y1) to (x2,y2). This type is available only for 2D plots.

PLOT_PLOTTYPE_FSTEPS Adjacent points are connected with two line segments, one from (x1,y1) to (x1,y2), and a second from (x1,y2) to (x2,y2). This type is available only for 2D plots.

PLOT_PLOTTYPE_HISTEPS This type is intended for plotting histograms. The point x1 is represented by a horizontal line from ((x0+x1)/2,y1) to ((x1+x2)/2,y1). Adjacent lines are connected with a vertical line from ((x1+x2)/2,y1) to ((x1+x2)/2,y2). This type is available only for 2D plots.

PLOT_PLOTTYPE_POINTS Markers are displayed at each data point.

PLOT_PLOTTYPE_LINESPOINTS Markers are displayed at each data point and connected with a line.

PLOT_PLOTTYPE_DOTS A small dot is displayed at each data point. The optional *point_type* argument effects only the color of the dot.

PLOT_PLOTTYPE_SURFACES Data points are meshed as a smooth surface. This is the default for 3D plots.

PLOT_PLOTTYPE_FILLEDCURVES The `filledcurves` plot type is only relevant to 2D plotting. Three variants are possible. The first two variants require two columns of input data, and may be further modified by the options listed below. The first variant, `closed`, treats the curve itself as a closed polygon. This is the default. The second variant is to fill the area between the curve and a given axis, a horizontal or vertical line, or a point. The third variant requires three columns of input data: the x coordinate and two y coordinates corresponding to two curves sampled at the same set of x coordinates; the area between the two curves is filled.

PLOT_PLOTTYPE_VECTORS For a 2D plot, this plot type draws a vector from (x,y) to (x+xdelta, y+ydelta). Thus it requires four columns of data. It also draws a small arrowhead at the end of the vector. A 3D plot of this plot type is similar, but requires six columns of data. It only works for Cartesian 3D plot. The `option` for this plot type is the same as that for the arrow style defined on page 11 for member function **CPlot**::**arrow**().

PLOT_PLOTTYPE_BOXES The boxes plot type is only relevant to 2-d plotting. It draws a box centered about the given x coordinate from the x axis (not the graph border) to the given y coordinate. The width of the box is obtained in one of three ways. If it is a data plot and the data file has a third column, this will be used to set the width of the box. If not, if a width has

been set using the member function **CPlot**::**boxWidth**(), this will be used. If neither of these is available, the width of each box will be calculated automatically so that it touches the adjacent boxes. The interior of the boxes is drawn based on the member function **CPlot**::**boxFill**(). By default, the the box is filled with the background color.

PLOT_PLOTTYPE_BOXERRORBARS The boxerrorbars plot type is only relevant to 2-d data plotting. It is a combination of the `boxes` **PLOT_PLOTTYPE_BOXES** and `yerrorbars` **PLOT_PLOTTYPE_YERRORBARS** plot types. The boxwidth will come from the fourth column if the y errors are in the form of `"ydelta"` and the boxwidth was not previously set equal to -2.0 by the member function **CPlot**::**boxWidth**() or from the fifth column if the y errors are in the form of `"ylow yhigh"`. The special case **CPlot**::**boxWidth**(-2.0) is for four-column data with y errors in the form `"ylow yhigh"`. In this case the boxwidth will be calculated so that each box touches the adjacent boxes. The width will also be calculated in cases where three-column data are used. The box height is determined from the y error in the same way as it is for the yerrorbars style—either from y-ydelta to y+ydelta or from ylow to yhigh, depending on how many data columns are provided. The interior of the boxes is drawn based on the specification by **CPlot**::**boxFill**().

PLOT_PLOTTYPE_BOXXYERRORBARS The `boxxyerrorbars` plot type is only relevant to 2-d data plotting. It is a combination of the `boxes` **PLOT_PLOTTYPE_BOXES** and `xyerrorbars` **PLOT_PLOTTYPE_XYERRORBARS** plot types. The box width and height are determined from the x and y errors in the same way as they are for the `xyerrorbars` plot type — either from xlow to xhigh and from ylow to yhigh, or from x-xdelta to x+xdelta and from y-ydelta to y+ydelta , depending on how many data columns are provided. The interior of the boxes is drawn based on the specification by **CPlot**::**boxFill**().

PLOT_PLOTTYPE_CANDLESTICKS The candlesticks plot type can be used for 2-d data plotting of financial data or for generating box-and-whisker plots of statistical data. Five columns of data are required; in order, these should be the x coordinate (most likely a date) and the opening, low, high, and closing prices. The symbol is a rectangular box, centered horizontally at the x coordinate and limited vertically by the opening and closing prices. A vertical line segment at the x coordinate extends up from the top of the rectangle to the high price and another down to the low. The vertical line will be unchanged if the low and high prices are interchanged.

The width of the rectangle can be controlled by the member function **CPlot**::**boxWidth**().

By default the vertical line segments have no crossbars at the top and bottom. If you want crossbars, which are typically used for box-and-whisker plots, then add the keyword `whiskerbars` to the `option` parameter of the function. By default these whiskerbars extend the full horizontal width of the candlestick, but you can modify this by specifying a fraction of the full width.

By default the rectangle is empty if (open $>$ close), and filled with three vertical bars if (close $>$ open). If filled-boxes support is present, then the rectangle can be colored by the member function **CPlot**::**boxFill**().

Note: To place additional symbols, such as the median value, on a box-and-whisker plot requires additional function call as shown in the example below.

```
plot.dataFile("candlesticks.dat", "using 1:3:2:6:5");
plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 0,
              "linetype 1 linewidth 2 whiskerbars 0.5");
plot.dataFile("candlesticks.dat", "using 1:4:4:4:4");
plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 1, "linetype -1 linewidth 2");
```

It assumed that the data in file `candlesticks.dat` contains the following entries

```
# X   Min   1stQuartile Median 3rdQuartile Max
  1   1.5   2           2.4    4           6.
  2   1.5   3           3.5    4           5.5
  3   4.5   5           5.5    6           6.5
  ...
```

The plot will have crossbars on the whiskers and crossbars are 50% of full width.

PLOT_PLOTTYPE_FINANCEBARS The `financebars` plot type is only relevant for 2-d data plotting of financial data. Five columns of data are required; in order, these should be the x coordinate (most likely a date) and the opening, low, high, and closing prices. The symbol is a vertical line segment, located horizontally at the x coordinate and limited vertically by the high and low prices. A horizontal tic on the left marks the opening price and one on the right marks the closing price. The length of these tics may be changed by **CPlot**::**barSize**(). The symbol will be unchanged if the high and low prices are interchanged.

PLOT_PLOTTYPE_XERRORBARS The `xerrorbars` plot type is only relevant to 2D data plotting. The `xerrorbars` is like `dots`, except that a horizontal error bar is also drawn. At each point (x,y), a line is drawn from (xlow,y) to (xhigh,y) or from (x-xdelta,y) to (x+xdelta,y), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

PLOT_PLOTTYPE_XYERRORBARS The `xyerrorbars` plot type is only relevant to 2D data plotting. The `xyerrorbars` is like `dots`, except that horizontal and vertical error bars are also drawn. At each point (x,y), lines are drawn from (x,y-ydelta) to (x,y+ydelta) and from (x-xdelta,y) to (x+xdelta,y) or from (x,ylow) to (x,yhigh) and from (xlow,y) to (xhigh,y), depending upon the number of data columns provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

If data in a file are provided in an unsupported mixed form, the option `using` filter for **CPlot**::dataFile() should be used to set up the appropriate form. For example, if the data are of the form (x,y,xdelta,ylow,yhigh), then you can use

```
plot.dataFile("datafile", "using 1:2:($1-$3):($1+$3):4:5");
plot.plotType(PLOT_PLOTTYPE_XYERRORBARS, plot.dataSetNum());
```

PLOT_PLOTTYPE_YERRORBARS The `yerrorbars` (or `errorbars`) plot type is only relevant to 2D data plotting. The `yerrorbars` is like `points`, except that a vertical error bar

is also drawn. At each point (x,y), a line is drawn from (x,y-ydelta) to (x,y+ydelta) or from (x,ylow) to (x,yhigh), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

PLOT_PLOTTYPE_XERRORLINES The `xerrorlines` plot type is only relevant to 2D data plotting. The `xerrorlines` is like `linespoints`, except that a horizontal error line is also drawn. At each point (x,y), a line is drawn from (xlow,y) to (xhigh,y) or from (x-xdelta,y) to (x+xdelta,y), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

PLOT_PLOTTYPE_XYERRORLINES The `xyerrorlines` plot type is only relevant to 2D data plotting. The `xyerrorlines` is like `linespoints`, except that a horizontal and vertical error lines are also drawn. At each point (x,y), lines are drawn from (x,y-ydelta) to (x,y+ydelta) and from (x-xdelta,y) to (x+xdelta,y) or from (x,ylow) to (x,yhigh) and from (xlow,y) to (xhigh,y), depending upon the number of data columns provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

If data in a file are provided in an unsupported mixed form, the option `using` filter for **CPlot**::dataFile() should be used to set up the appropriate form. For example, if the data are of the form (x,y,xdelta,ylow,yhigh), then you can use

```
plot.dataFile("datafile", "using 1:2:($1-$3):($1+$3):4:5");
plot.plotType(PLOT_PLOTTYPE_XYERRORLINES, plot.dataSetNum());
```

PLOT_PLOTTYPE_YERRORLINES The `yerrorlines` (or `errorlines`) plot type is only relevant to 2D data plotting. The `yerrorlines` is like `linespoints`, except that a vertical error line is also drawn. At each point (x,y), a line is drawn from (x,y-ydelta) to (x,y+ydelta) or from (x,ylow) to (x,yhigh), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

num The data set to which the plot type applies.

line_type An integer index representing the line type for drawing. Use the same value for different curves so that each curve with the same color and style.

line_width A scaling factor for the line width. The line width is *line_width* multiplied by the default width.

point_type An integer index representing the desired point type.

point_size A scaling factor for the size of the point used. The point size is *point_size* multiplied by the default size.

option An option string for a plot type to fine tune the plot.

1. The `option` for the plot type **PLOT_PLOTTYPE_VECTORS** is the same as that for the arrow style defined on page 11 for member function **CPlot**::**arrow**().
2. The `option` for the plot type **PLOT_PLOTTYPE_CANDLESTICKS** is as follows.

```
{ {linetype | lt <line_type>}
  {linewidth | lw <line_width}
  {whiskerbars [fraction_value]}
}
```

It specifies line type and line width. The `whiskerbars` extend the full horizontal width of the candlestick. The optional `fraction_value` in the range [0, 1.0] specifies a fraction of the full width of whiskerbars.

3. The `option` for the plot type **PLOT_PLOTTYPE_FILLEDCURVES** is as follows.

```
{ [closed | {above | below} {x1 | x2 | y1 | y2}[=<a>] | xy=<x>,<y>]
  {linetype | lt <line_type>}
}
```

The option `linetype` can be used to change the color for the filled area. For example,

```
plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, num, "y1=0 linetype 1");
```

The first two plot variants for **PLOT_PLOTTYPE_FILLEDCURVES** can be further modified by the options

```
closed   ... just filled closed curve,
x1       ... x1 axis,
x2       ... x2 axis, etc for y1 and y2 axes,
y1=0     ... line y=0 (at y1 axis) ie parallel to x1 axis,
y2=42    ... line y=42 (at y2 axis) ie parallel to x2, etc,
xy=10,20 ... point 10,20 of x1,y1 axes (arc-like shape).
```

An example of filling the area between two input curves using three columns of data is as follows.

```
plot.dataFile("datafile", "using 1:2:3");
plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, plot.dataSetNum());
```

The `above` and `below` in the form

```
above {x1|x2|y1|y2}=<val>
below {x1|x2|y1|y2}=<val>
```

limit the filled area to one side of the bounding line or curve.

If the values of `<a>, <x>, <y>` are out of the drawing boundary, then they are moved to the graph boundary. Then the actually filled area in the case of option `xy=<x>,<y>` will depend on the x-range and y-range.

Description

Set the desired plot type for a previously added data set. For 3D plots, only **PLOT_PLOTTYPE_LINES**, **PLOT_PLOTTYPE_IMPULSES**, **PLOT_PLOTTYPE_POINTS**, and **PLOT_PLOTTYPE_LINESPOINTS** are valid. If other types are specified, **PLOT_PLOTTYPE_POINTS** is used.

Some 2D plot types need data with with more than two columns. If the data are in a file, the `using` option of **CPlot**::**dataFile**() can be used to set up the correct columns for the plot type you want. In this discussion, "column" will be used to refer to a column in a data file, an entry in the `using` list, or a column in a two-dimensional array. The data for plotting in a two-dimensonal array can be added to an instance of **CPlot** class by the member function **CPlot**::**data**().

For three columns data, only `xerrorbars, yerrorbars, xerrorlines, yerrorlines, boxes, boxerrorbars`, and `filledcurves` are allowed. If other plot type is used, the type will be changed to `yerrorbars`.

For four columns, only `xerrorbars, yerrorbars, xyerrorbars, xerrorlines, yerrorlines, xyerrorlines, boxxyerrorbars`, and `boxerrorbars` are allowed. An illegal plot type will be changed to `yerrorbars`.

Five-column data allow only the `boxerrorbars, financebars`, and `candlesticks` plot types. An illegal style will be changed to `boxerrorbars` before plotting.

Six- and seven-column data only allow the `xyerrorbars, xyerrorlines`, and `boxxyerrorbars` plot types. Illegal styles will be changed to `xyerrorbars` before plotting.

Numbering of the data sets starts with zero. New plot types replace previously specified types. The line style and/or marker type for the plot are selected automatically, unless the appropriate combination of *line_type*, *line_width*, *point_type*, and *point_size* are specified. The *line_type* is an optional argument specifying an index for the line type used for drawing the line. The line type varies depending on the terminal type used (see **CPlot**::**outputType**). Typically, changing the line type will change the color of the line or make it dashed or dotted. All terminals support at least six different line types. By default, the line type is 1. The *line_width* is an optional argument used to specify the line width. The line width is *line_width* multiplied by the default width. Typically the default width is one pixel. *point_type* is an optional argument used to change the appearance (color and/or marker type) of a point. It is specified with an integer representing the index of the desired point type. All terminals support at least six different point types. *point_size* is an optional argument used to change the size of the point. The point size is *point_size* multiplied by the default size. If *point_type* and *point_size* are set to zero or a negative number, a default value is used.

Portability
The *line_width* and *point_size* options is not supported by all terminal types.

For 3D plots on some systems with output type set to `postscript` (see **CPlot**::**outputType**()), data may not be displayed for **PLOT_PLOTTYPE_DOTS**.

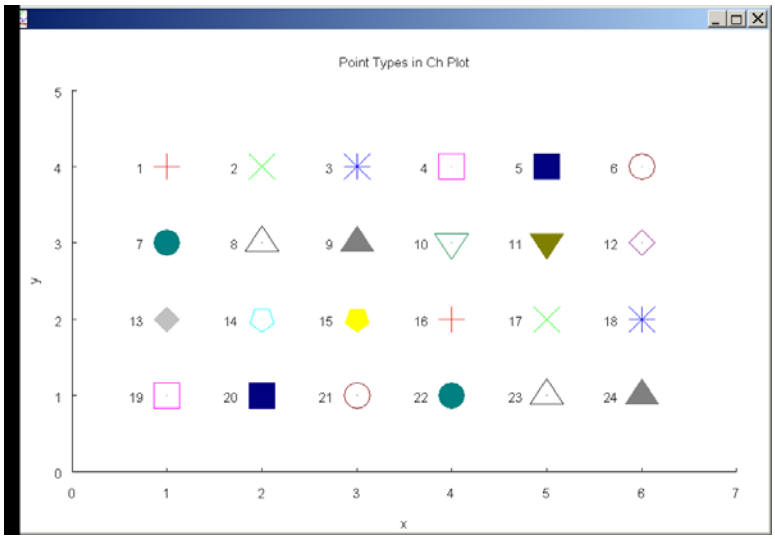
Example 1
This example shows some of the different point types for the default X-window and the `postscript` terminal types (see **CPlot**::**outputType**). In this example the points have a point size of five times the default. The appearance of points for different terminal types may be different.

```
/* File: plotType4.ch */
#include <chplot.h>

int main() {
```

```
    double x, y;
    string_t text;
    int datasetnum=0, point_type = 1, point_size = 5;
    class CPlot plot;

    plot.axisRange(PLOT_AXIS_X, 0, 7);
    plot.axisRange(PLOT_AXIS_Y, 0, 5);
    plot.title("Point Types in Ch Plot");
    for (y = 4; y >= 1; y--) {
        for (x = 1; x <= 6; x++) {
            sprintf(text, "%d", point_type);
            plot.point(x, y, 0);
            plot.plotType(PLOT_PLOTTYPE_POINTS, datasetnum);
            plot.pointType(datasetnum,  point_type, point_size);
            plot.text(text, PLOT_TEXT_RIGHT, x-.25, y, 0);
            datasetnum++; point_type++;
        }
    }
    plot.plotting();
}
```

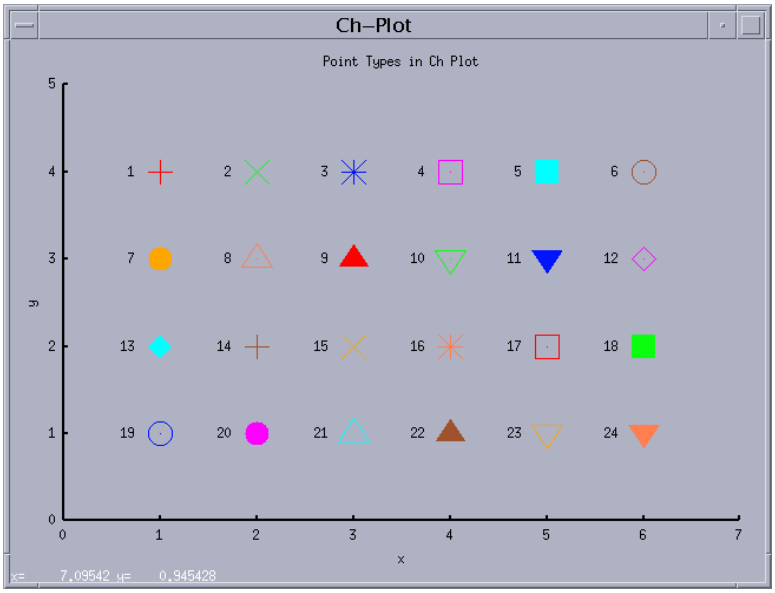
Output

Output displayed in Window.



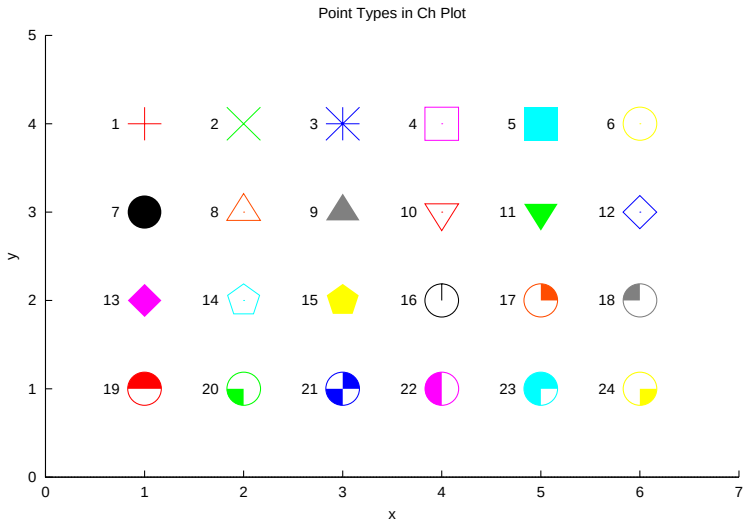
Output

Output displayed in X-window.



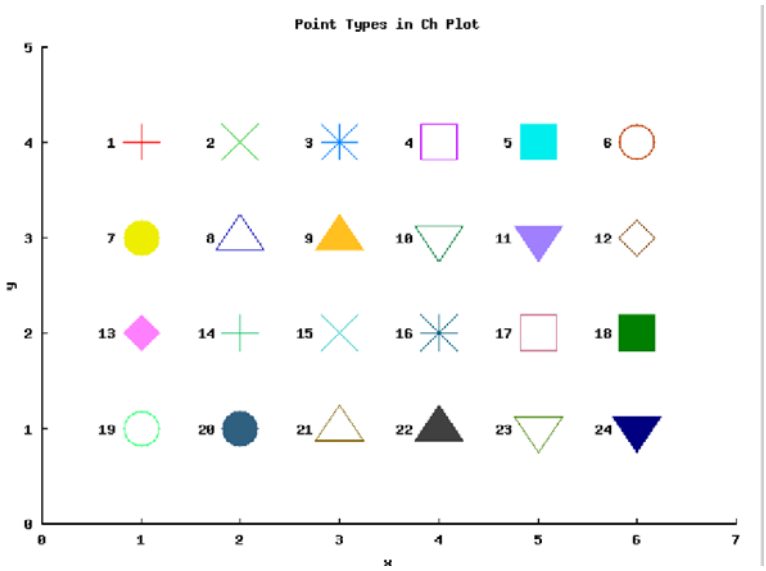
Output

Output as a postscript file.



Output

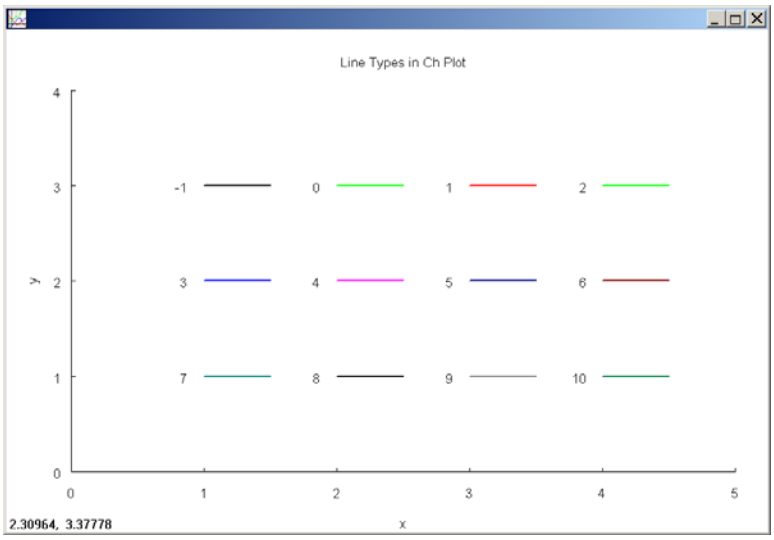
Output as a PNG file.



Example 2
This example shows some of the different line types for the `postscript` terminal type (see **CPlot**::**outputType**). The appearance of lines for different terminal types may be different.

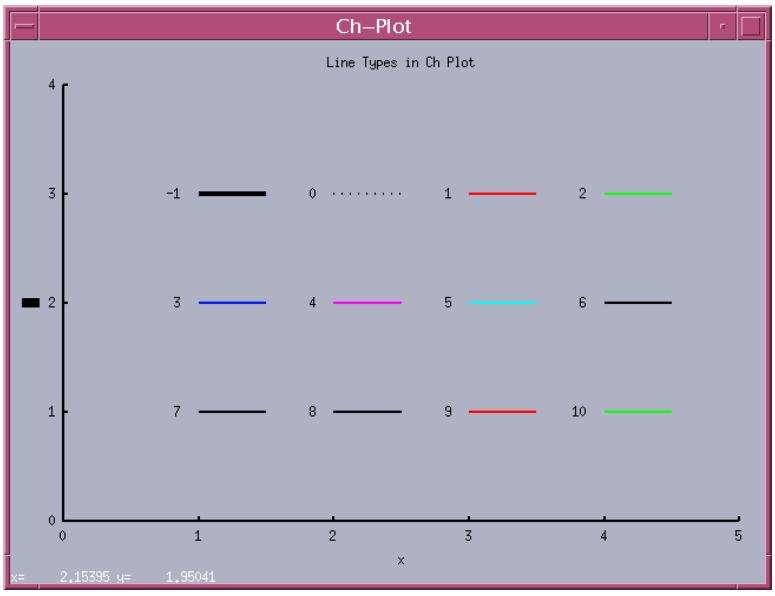
```
/* File: plotType5.ch */
#include <chplot.h>

int main() {
    double x, y, xx[2], yy[2];
    string_t text;
    int line_type = -1, line_width = 2, datasetnum = 0;
    class CPlot plot;

    plot.axisRange(PLOT_AXIS_X, 0, 5);
    plot.axisRange(PLOT_AXIS_Y, 0, 4);
    plot.ticsRange(PLOT_AXIS_Y, 1, 0, 4);
    plot.title("Line Types in Ch Plot");
    for (y = 3; y >= 1; y--) {
        for (x = 1; x <= 4; x++) {
            sprintf(text, "%d", line_type);
            lindata(x, x+.5, xx);
            lindata(y, y, yy);
            plot.data2D(xx, yy);
            plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
            plot.lineType(datasetnum, line_type, line_width);
            plot.text(text, PLOT_TEXT_RIGHT, x-.125, y, 0);
            datasetnum++;
            line_type++;
         }
    }
    plot.plotting();
}
```

Output
Output displayed in Window.



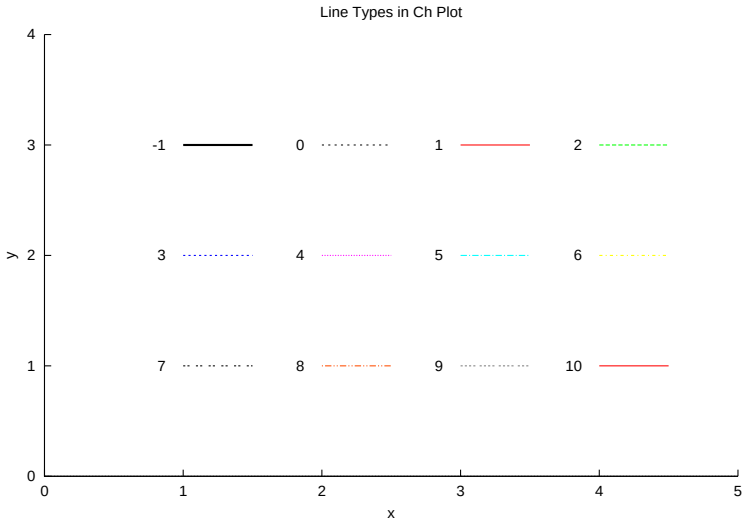
Output

Output displayed in X-window.



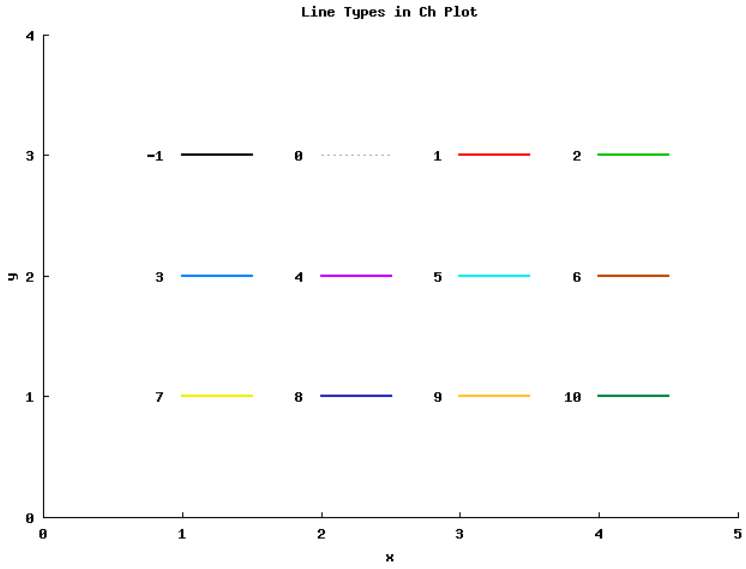
Output

Output as a postscript file.



Output

Output as a PNG file.



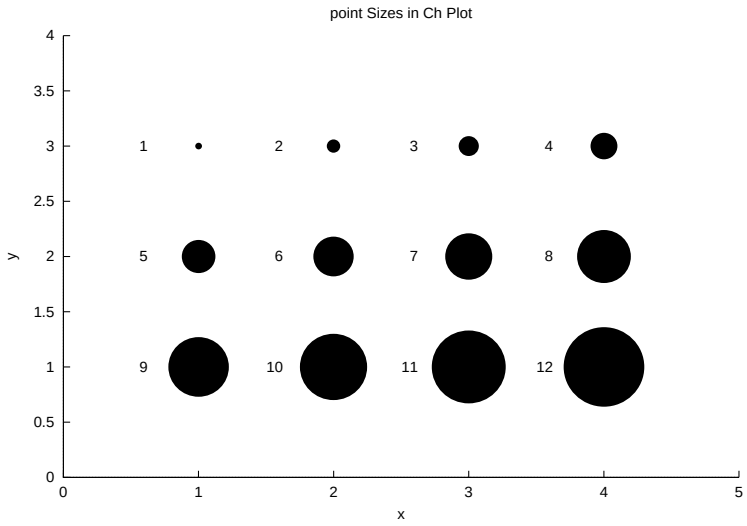
Example 3

This example shows some of the different point sizes for the `postscript` terminal type (see **CPlot**::**outputType**). The appearance of points for different terminal types may be different.

```
/* File: plotType6.ch */
#include <chplot.h>

int main() {
    double x, y;
    string_t text;
    int datasetnum=0, point_type = 7, point_size = 1;
    class CPlot plot;
```

```
    plot.axisRange(PLOT_AXIS_X, 0, 5);
    plot.axisRange(PLOT_AXIS_Y, 0, 4);
    plot.title("point Sizes in Ch Plot");
    for (y = 3; y >= 1; y--) {
        for (x = 1; x <= 4; x++) {
            sprintf(text, "%d", point_size);
            plot.point(x, y, 0);
            plot.plotType(PLOT_PLOTTYPE_POINTS, datasetnum);
            plot.pointType(datasetnum,  point_type, point_size);
            plot.text(text, PLOT_TEXT_RIGHT, x-.375, y, 0);
            datasetnum++; point_size++;
        }
    }
    plot.plotting();
}
```

Output

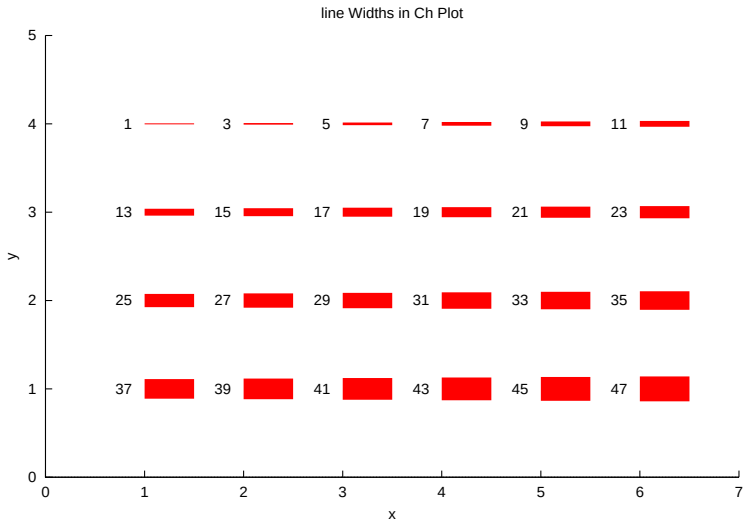


Example 4
This example shows some of the different line sizes for the `postscript` terminal type (see **CPlot**::**outputType**). The appearance of lines for different terminal types may be different.

```
/* File: plotType7.ch */
#include <chplot.h>

int main() {
    double x, y, xx[2], yy[2];
    string_t text;
    int line_type = 1, line_width = 1, datasetnum = 0;
    class CPlot plot;

    plot.axisRange(PLOT_AXIS_X, 0, 7);
    plot.axisRange(PLOT_AXIS_Y, 0, 5);
    plot.title("line Widths in Ch Plot");
    for (y = 4; y >= 1; y--) {
        for (x = 1; x <= 6; x++) {
            sprintf(text, "%d", line_width);
            lindata(x, x+.5, xx);
            lindata(y, y, yy);
```

```
            plot.data2D(xx, yy);
            plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
            plot.lineType(datasetnum, line_type, line_width);
            plot.text(text, PLOT_TEXT_RIGHT, x-.125, y, 0);
            datasetnum++;
            line_width+=2;
        }
    }
    plot.plotting();
}
```

Output

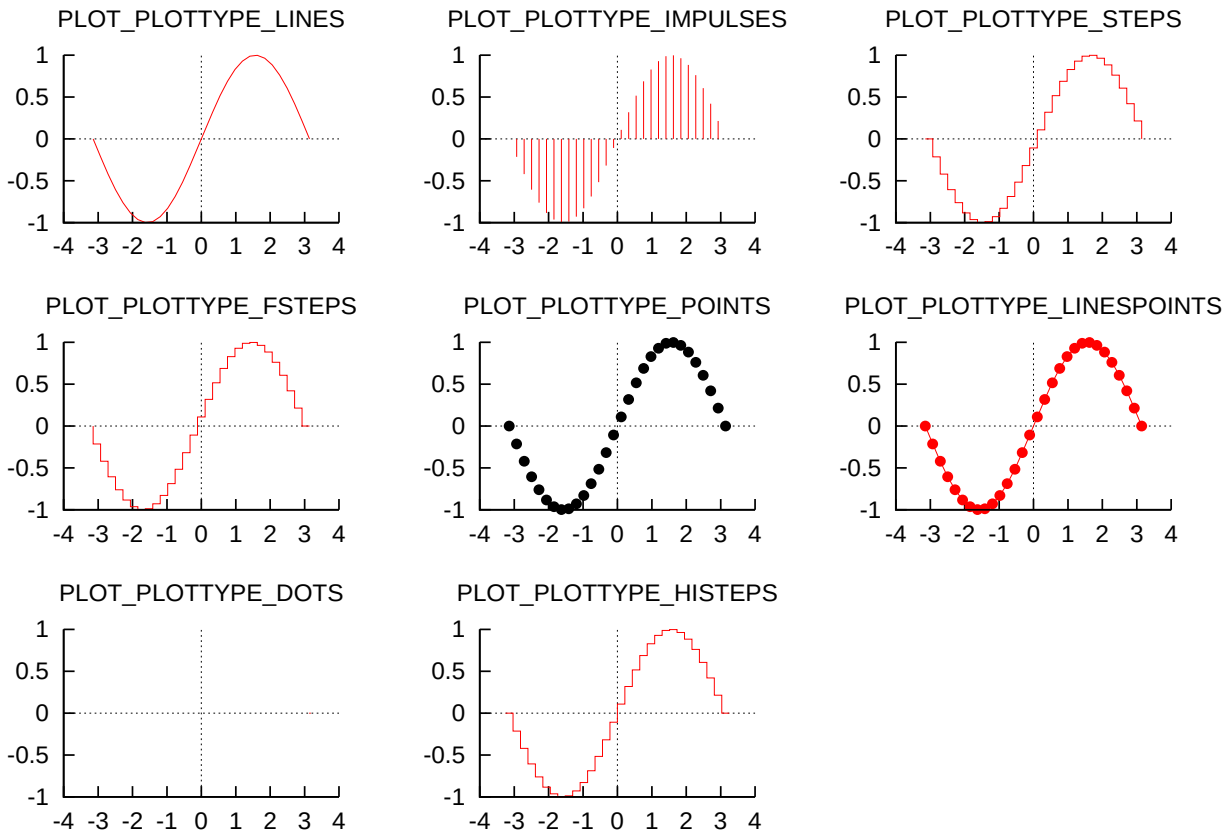


Example 5

```
/* File: plotType1.ch */
#include <chplot.h>
#include <math.h>

int main() {
    class CPlot subplot, *spl;
    array double x1[30], y1[30];
    int line_type = 1, line_width = 1;
    int point_type = 7, point_size = 1;

    lindata(-M_PI, M_PI, x1);
    y1 = sin(x1);
    subplot.subplot(3,3);
    spl = subplot.getSubplot(0,0);
    spl->data2D(x1, y1);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_LINES, 0);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_LINES");
    spl = subplot.getSubplot(0,1);
    spl->data2D(x1, y1);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_IMPULSES, 0);
```

```
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_IMPULSES");
    spl = subplot.getSubplot(0,2);
    spl->data2D(x1, y1);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_STEPS, 0);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_STEPS");
    spl = subplot.getSubplot(1,0);
    spl->data2D(x1, y1);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_FSTEPS, 0);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_FSTEPS");
    spl = subplot.getSubplot(1,1);
    spl->data2D(x1, y1);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_POINTS, 0);
    spl->pointType(0, point_type, point_size);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_POINTS");
    spl = subplot.getSubplot(1,2);
    spl->data2D(x1, y1);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_LINESPOINTS, 0);
    spl->lineType(0, line_type, line_width);
    spl->pointType(0, point_type, point_size);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_LINESPOINTS");
    spl = subplot.getSubplot(2,0);
    spl->data2D(x1, y1);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_DOTS, 0);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_DOTS");
    spl = subplot.getSubplot(2,1);
    spl->data2D(x1, y1);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_HISTEPS, 0);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_HISTEPS");
    subplot.plotting();
}
```

Output

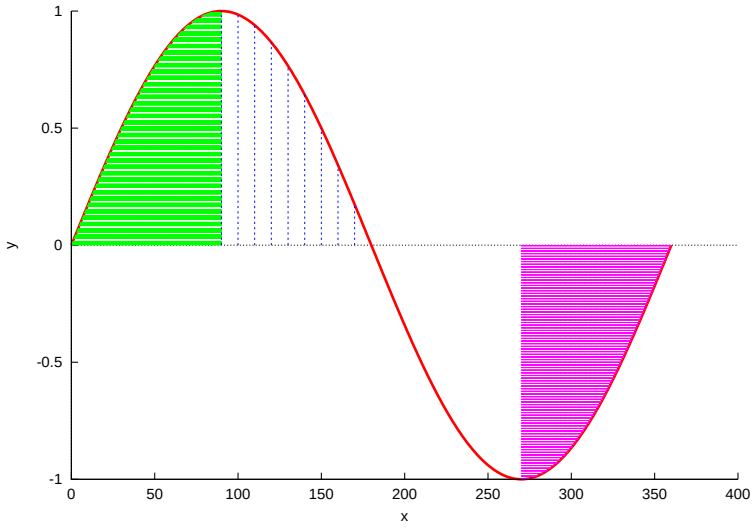


Example 6

```
/* File: plotType2.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 360,        // number of data points
        numpts=10;              // number of data points
    array double x0[numpoints], y0[numpoints];
    array double x1[numpoints], y1[numpoints];
    array double x2[numpts], y2[numpts];
    array double x3[numpoints], y3[numpoints];
    int line_type=1, line_width =5;
    class CPlot plot;

    lindata(0, 360, x0);   // assign x0 with -10 <= x0 <= 360 linearly
    y0 = sin(x0*M_PI/180);  // array y0 is sine of array x0, element-wise
    lindata(0, 90, x1);    // assign x1 with 0 <= x1 <= 90 linearly
    y1 = sin(x1*M_PI/180);  // array y1 is sine of array x1, element-wise
    lindata(90, 180, x2);  // assign x2 with 90 <= x2 <= 180 linearly
    y2 = sin(x2*M_PI/180);  // array y2 is sine of array x2, element-wise
    lindata(270, 360, x3); // assign x3 with 90 <= x3 <= 180 linearly
    y3 = sin(x3*M_PI/180);  // array y3 is sine of array x3, element-wise
    plot.data2D(x0, y0);
    plot.data2D(x1, y1);
    plot.data2D(x2, y2);
    plot.data2D(x3, y3);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);       // line for (x,y)
```

```
    plot.lineType(0, line_type, line_width);
    plot.plotType(PLOT_PLOTTYPE_IMPULSES, 1);     // impulse plot for (x1, y1)
    plot.plotType(PLOT_PLOTTYPE_IMPULSES, 2);     // impulse plot for (x2, y2)
    plot.plotType(PLOT_PLOTTYPE_IMPULSES, 3);     // impulse plot for (x3, y3)
    plot.plotting();                              // get the plotting job done
}
```

Output



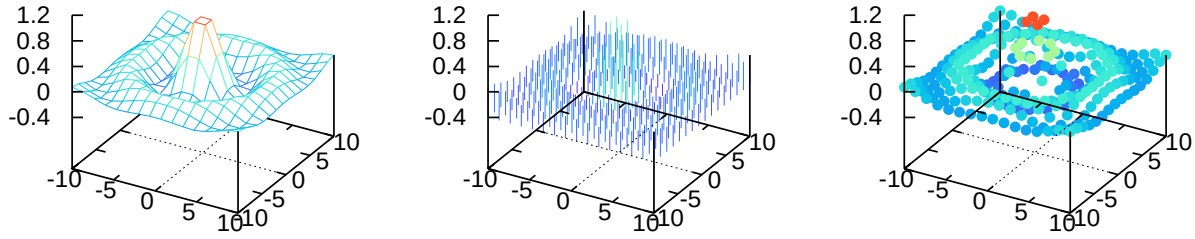
Example 7

```
/* File: plotType3.ch */
#include <chplot.h>
#include <math.h>

int main() {
    int line_type = 1, line_width = 1;
    int point_type = 7, point_size = 1;
    double x[16], y[16], z[256];
    double r;
    int i, j;
    class CPlot subplot, *spl;

    lindata(-10, 10, x);
    lindata(-10, 10, y);
    for(i=0; i<16; i++) {
        for(j=0; j<16; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[16*i+j] = sin(r)/r;
        }
    }
    subplot.subplot(2,3);
    spl = subplot.getSubplot(0,0);
    spl->data3D(x, y, z);
    spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
    spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
    spl->axisRange(PLOT_AXIS_XY, -10, 10);
    spl->ticsRange(PLOT_AXIS_XY, 5, -10, 10);
    spl->plotType(PLOT_PLOTTYPE_LINES, 0);
    spl->colorBox(PLOT_OFF);
```

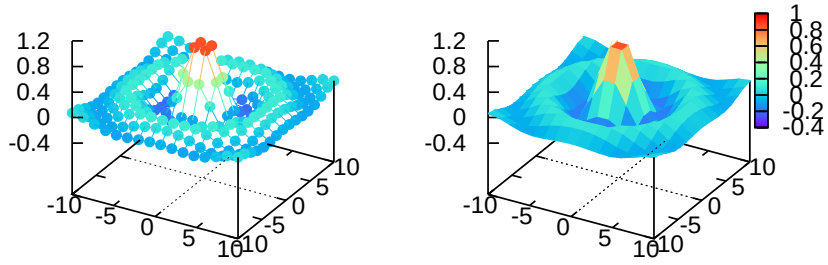
```
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->title("PLOT_PLOTTYPE_LINES");

    spl = subplot.getSubplot(0,1);
    spl->data3D(x, y, z);
    spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
    spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
    spl->axisRange(PLOT_AXIS_XY, -10, 10);
    spl->ticsRange(PLOT_AXIS_XY, 5, -10, 10);
    spl->plotType(PLOT_PLOTTYPE_IMPULSES, 0);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->title("PLOT_PLOTTYPE_IMPULSES");

    spl = subplot.getSubplot(0,2);
    spl->data3D(x, y, z);
    spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
    spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
    spl->axisRange(PLOT_AXIS_XY, -10, 10);
    spl->ticsRange(PLOT_AXIS_XY, 5, -10, 10);
    spl->plotType(PLOT_PLOTTYPE_POINTS, 0);
    spl->pointType(0, point_type, point_size);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->title("PLOT_PLOTTYPE_POINTS");

    spl = subplot.getSubplot(1,0);
    spl->data3D(x, y, z);
    spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
    spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
    spl->axisRange(PLOT_AXIS_XY, -10, 10);
    spl->ticsRange(PLOT_AXIS_XY, 5, -10, 10);
    spl->plotType(PLOT_PLOTTYPE_LINESPOINTS, 0);
    spl->lineType(0, line_type, line_width);
    spl->pointType(0, point_type, point_size);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->title("PLOT_PLOTTYPE_LINESPOINTS");

    spl = subplot.getSubplot(1,1);
    spl->data3D(x, y, z);
    spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
    spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
    spl->axisRange(PLOT_AXIS_XY, -10, 10);
    spl->ticsRange(PLOT_AXIS_XY, 5, -10, 10);
    spl->plotType(PLOT_PLOTTYPE_SURFACES, 0);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->title("PLOT_PLOTTYPE_SURFACES");
    subplot.plotting();
}
```

Output

PLOT_PLOTTYPE_LINES PLOT_PLOTTYPE_IMPULSES PLOT_PLOTTYPE_POINTS



PLOT_PLOTTYPE_LINESPOINTS PLOT_PLOTTYPE_SURFACES



Example 8
This example illustrates the plot type **PLOT_PLOTTYPE_VECTORS**. The arrow style is defined based on the specification in **CPlot**::**arrow**() on page 11.

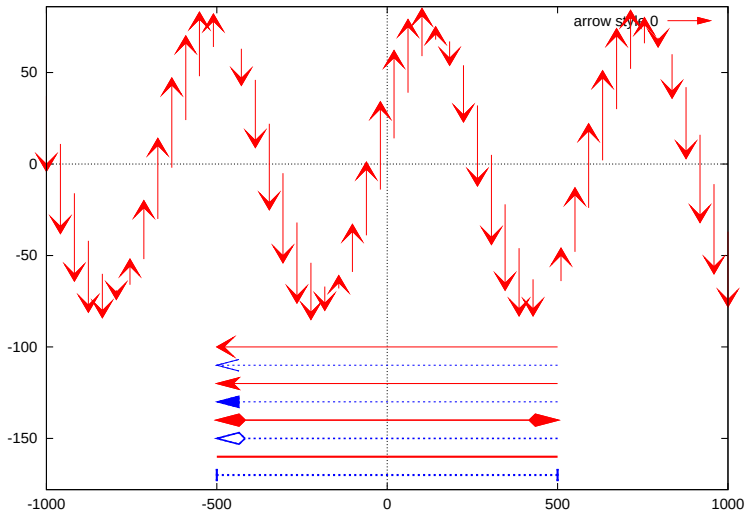
```
/* File: plotType_v.ch */
#include <chplot.h>

char *arrowstyle[] = {
             "head filled size screen 0.025,30,45 linetype 1 linewidth 1",
             "head nofilled size screen 0.03,15 linetype 3 linewidth 1",
             "head filled size screen 0.03,15,45 linetype 1 linewidth 2",
             "head filled size screen 0.03,15 linetype 3 linewidth 2",
             "heads filled size screen 0.03,15,135 linetype 1 linewidth 3",
             "head empty size screen 0.03,15,135 linetype 3 linewidth 3",
             "nohead linetype 1 linewidth 4",
             "heads size screen 0.008,90 linetype 3 linewidth 4"
};

int main () {
    class CPlot plot;
    int i;

    plot.label(PLOT_AXIS_X, "");
    plot.label(PLOT_AXIS_Y, "");
    plot.axisRange(PLOT_AXIS_X, -1000, 1000);
    plot.axisRange(PLOT_AXIS_Y,-178, 86);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);
```

```
    for(i=0; i<sizeof(arrowstyle)/sizeof(char*); i++) {
      plot.arrow(-500,-100-10*i,0, 500,-100-10*i,0, arrowstyle[i]);
    }
    plot.dataFile("arrowstyle.dat", "using 1:2:(0):3");
    plot.plotType(PLOT_PLOTTYPE_VECTORS, 0, arrowstyle[0]);
    plot.legend("arrow style 0", 0);
    plot.plotting();
}
```

The contents of data file `arrowstyle.dat` in Example 8

```
-1000    37      -41
-959     11      -49
-918     -16     -48
...
```

Output



Example 9
This example illustrates the plot type **PLOT_PLOTTYPE_CANDLESTICKS**.

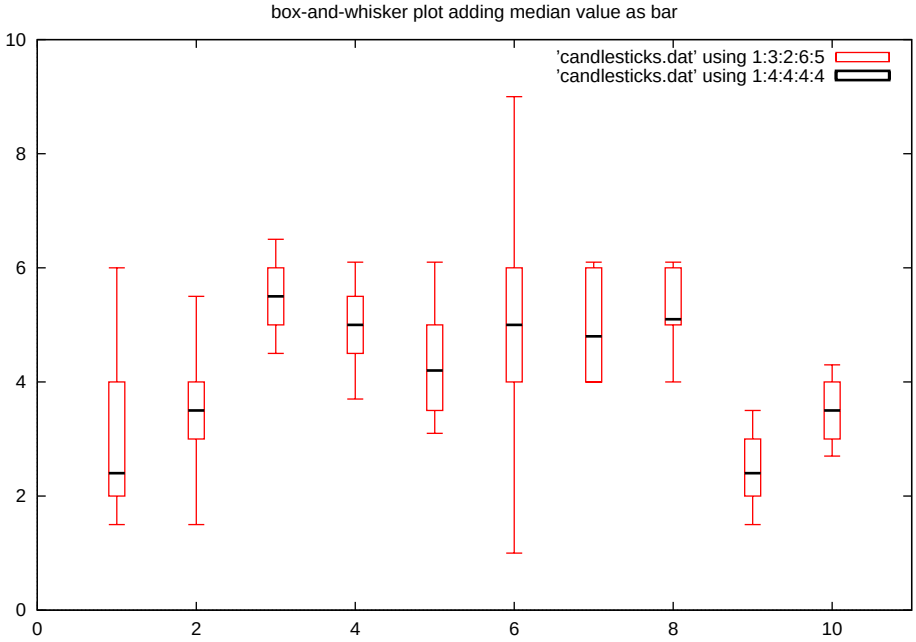
```
/* File: plotType_cs.ch */
/* File: plotTtype_cs.ch to process data in candlesticks.dat */
#include <chplot.h>

int main() {
    class CPlot plot;

    plot.label(PLOT_AXIS_X, "");
    plot.label(PLOT_AXIS_Y, "");
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot.title("box-and-whisker plot adding median value as bar");
    plot.axisRange(PLOT_AXIS_X, 0, 11);
    plot.axisRange(PLOT_AXIS_Y, 0, 10);
    plot.dataFile("candlesticks.dat", "using 1:3:2:6:5");
    plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 0, "linetype 1 linewidth 2 whiskerbars");
    plot.boxFill(0, PLOT_BOXFILL_EMPTY);
```

```
    plot.legend("'candlesticks.dat' using 1:3:2:6:5", 0);
    plot.dataFile("candlesticks.dat", "using 1:4:4:4:4");
    plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 1, "linetype -1 linewidth 2");
    plot.legend("'candlesticks.dat' using 1:4:4:4:4", 1);
    plot.boxWidth(0.2);
    plot.plotting();
    return 0;
}
```

The contents of data file `candlesticks.dat` in Example 9

```
1       1.5     2       2.4     4       6.
2       1.5     3       3.5     4       5.5
3       4.5     5       5.5     6       6.5
...
```

Output



Example 10
This example illustrates the plot type **PLOT_PLOTTYPE_FINANCEBARS**.

```
/* File: plotType_fb.ch */
/* File: plotType_fb.ch to process data in finance.dat

  data and indicators in finance.dat
  # data file layout:
  # date, open, high, low, close, volume,
  # 50-day moving average volume, Intraday Intensity,
  # 20-day moving average close,
  # upper Bollinger Band, lower Bollinger Band
*/

#include <chplot.h>
```

```
int main() {
    class CPlot subplot, *plot1, *plot2;

    subplot.subplot(2,1);
    subplot.title("Finance Bar");
    plot1 = subplot.getSubplot(0,0);
    plot1->label(PLOT_AXIS_X, "");
    plot1->label(PLOT_AXIS_Y, "price");
    plot1->border(PLOT_BORDER_ALL, PLOT_ON);
    plot1->ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot1->dataFile("finance.dat", "using 0:2:3:4:5");
    plot1->plotType(PLOT_PLOTTYPE_FINANCEBARS, 0);
    plot1->axisRange(PLOT_AXIS_X, 50, 253);
    plot1->axisRange(PLOT_AXIS_Y, 75, 105);
    plot1->grid(PLOT_ON);
    plot1->ticsPosition(PLOT_AXIS_X, 66, 87, 109, 130, 151, 174, 193, 215, 235);
    plot1->ticsPosition(PLOT_AXIS_Y, 105, 100, 95, 90, 85, 80);
    plot1->ticsFormat(PLOT_AXIS_X, "");
    plot1->scaleType(PLOT_AXIS_Y, PLOT_SCALETYPE_LOG);
    plot1->dataFile("finance.dat", "using 0:9");
    plot1->dataFile("finance.dat", "using 0:10");
    plot1->dataFile("finance.dat", "using 0:11");
    plot1->dataFile("finance.dat", "using 0:8");
    plot1->axes(plot1->dataSetNum(), "x1y2");
    plot1->text("Courtesy of Bollinger Capital", PLOT_TEXT_CENTER, 83, 76.7, 0);
    plot1->text("www.BollingerBands.com", PLOT_TEXT_CENTER, 83, 75.7, 0);
    plot1->size(1, 0.7);           // 70% for the top plot
    plot1->margins(9, -1, -1, 0);  // left margin is 9, bottom margin is 0
    plot1->origin(0, 0.3);         // origin for Y is 0.3
    plot1->text("Acme Widgets", PLOT_TEXT_CENTER, 150, 102, 0);

    plot2 = subplot.getSubplot(1,0);
    plot2->label(PLOT_AXIS_X, "");
    plot2->label(PLOT_AXIS_Y, "volume (x10000)");
    plot2->dataFile("finance.dat", "using 0:($6/10000)");
    plot2->plotType(PLOT_PLOTTYPE_IMPULSES, 0);
    plot2->dataFile("finance.dat", "using 0:($7/10000)");
    plot2->border(PLOT_BORDER_ALL, PLOT_ON);
    plot2->ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot2->grid(PLOT_ON);
    plot2->axisRange(PLOT_AXIS_X, 50, 253);
    plot2->ticsRange(PLOT_AXIS_Y, 500);
    plot2->ticsFormat(PLOT_AXIS_Y, "%1.0f");
    plot2->ticsPosition(PLOT_AXIS_X, 66, 87, 109, 130, 151, 174, 193, 215, 235);
    /* use the code below for X-label for C++ or Ch */
    /*
      plot2->ticsLabel(PLOT_AXIS_X, "6/03", 66);
      plot2->ticsLabel(PLOT_AXIS_X, "7/03", 87);
      plot2->ticsLabel(PLOT_AXIS_X, "8/03", 109);
      plot2->ticsLabel(PLOT_AXIS_X, "9/03", 130);
      plot2->ticsLabel(PLOT_AXIS_X, "10/03", 151);
      plot2->ticsLabel(PLOT_AXIS_X, "11/03", 174);
      plot2->ticsLabel(PLOT_AXIS_X, "12/03", 193);
      plot2->ticsLabel(PLOT_AXIS_X, "1/04", 215);
      plot2->ticsLabel(PLOT_AXIS_X, "2/04", 235);
    */
    plot2->size(1, 0.3);          // 30% for the bottom plot
    plot2->margins(9, -1, 0, -1); // left margin is 9, top margin is 0
    subplot.plotting();
```

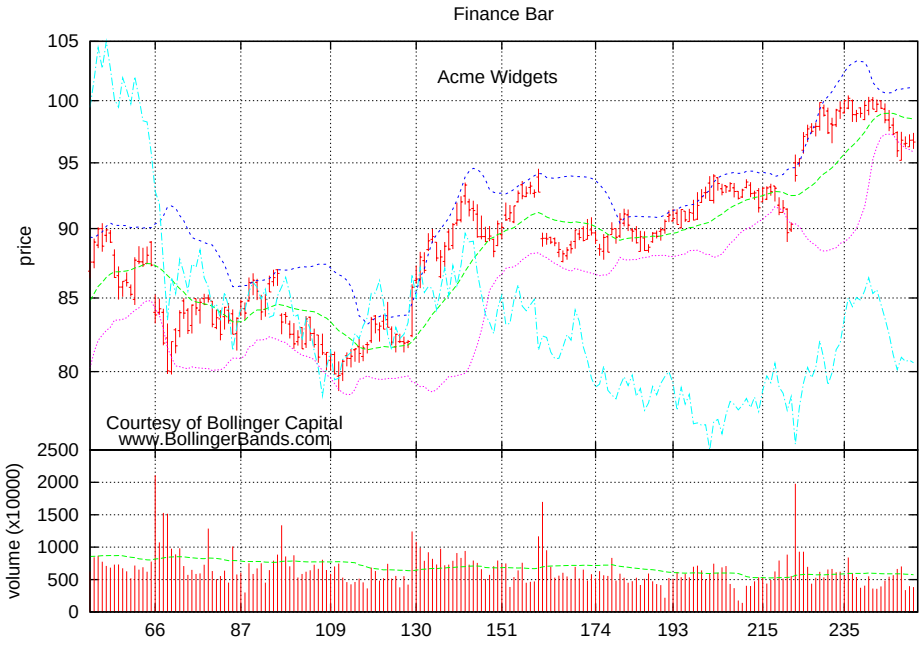
```
    return 0;
}
```

The contents of data file `finance.dat` in Example 10

```
# data and indicators in finance.dat
# data file layout:
# date, open, high, low, close, volume,
# 50-day moving average volume, Intraday Intensity,
# 20-day moving average close,
# upper Bollinger Band, lower Bollinger Band
2/27/2003 77.9  78.59 76.75 77.28 9927900 0 -4208566 0 0 0
2/28/2003 78.15 78.47 77    77.95 6556100 0 -2290796 0 0 0
3/3/2003  78.6  79    77.12 77.33 6618300 0 -7430539 0 0 0
...
```

This data file is also used to plot stock prices in Example 14 below.

Output



Example 11

This example illustrates the plot type **PLOT_PLOTTYPE_YERRORBARS**.

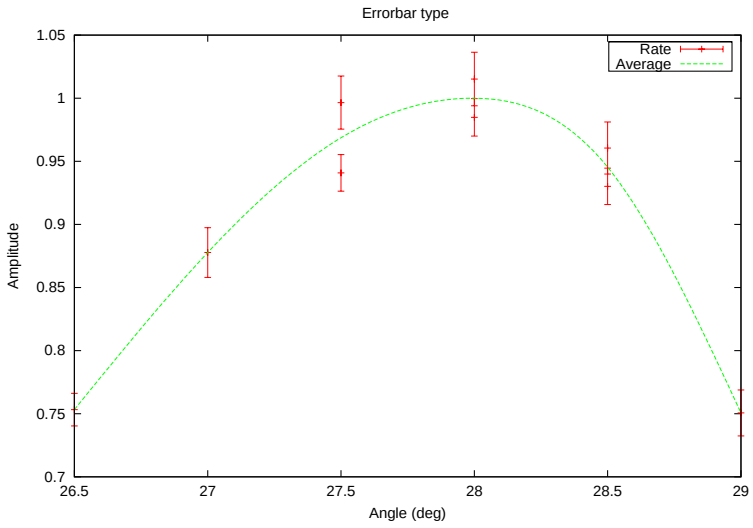
```
/* File: plotType_eb.ch */
#include<chplot.h>

int main() {
   CPlot plot;

   plot.title("Errorbar type ");
   plot.label(PLOT_AXIS_X, "Angle (deg)");
   plot.label(PLOT_AXIS_Y, "Amplitude");
```

```
   plot.border(PLOT_BORDER_ALL, PLOT_ON);
   plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);
   plot.legendOption("box");
   plot.dataFile("big_peak.dat");
   plot.plotType(PLOT_PLOTTYPE_YERRORBARS, 0);
   plot.legend("Rate", 0);
   plot.dataFile("big_peak.dat");
   plot.smooth(1, "csplines");
   plot.legend("Average", 1);
   plot.barSize(1.0);
   plot.plotting();
}
```

The contents of data file `big_peak.dat` **in Example 11**

```
26.500000 0.753252 0.012953
27.000000 0.877710 0.019712
27.500000 0.996531 0.021018
...
```

Output

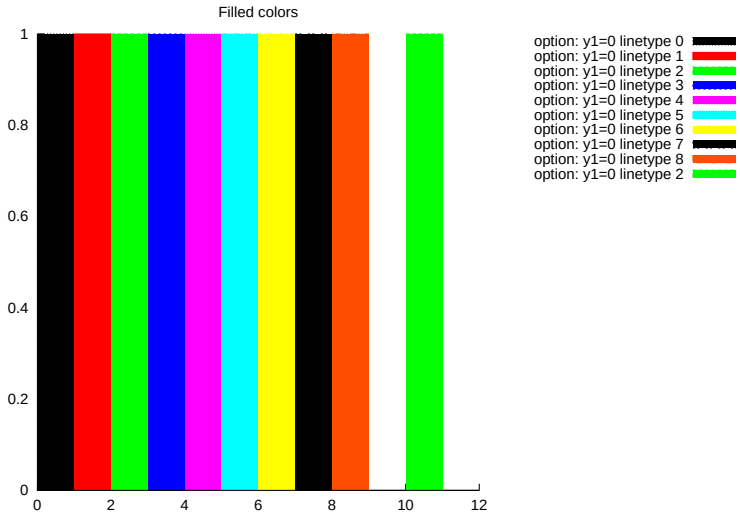


Example 12
This example illustrates the plot type **PLOT_PLOTTYPE_FILLEDCURVES**.

```
/* File: filledcolors.ch */
#include<math.h>
#include<chplot.h>

#define N 100

double func(double x) {
   double y;

   y = 1;
   return y;
}
```

```
int main() {
   CPlot plot;
   double x0, xf;
   int i;
   char option[64], legend[64];

   plot.title("Filled colors");
   plot.legendOption("outside");
   plot.label(PLOT_AXIS_X, "");
   plot.label(PLOT_AXIS_Y, "");
   plot.axisRange(PLOT_AXIS_X, 0, 12);
   plot.axisRange(PLOT_AXIS_Y, 0, 1);

   for(i=0; i<9; i++) {
     x0 = i;
     xf = i+1;
     sprintf(option, "y1=0 linetype %d", i);
     sprintf(legend, "option: %s", option);
     plot.func2D(x0, xf, N, func);
     plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, i, option);
     plot.legend(legend, i);
   }
   x0 = i+1;
   xf = i+2;
   plot.func2D(x0, xf, N, func);
   plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, 9, "y1=0 linetype 2");
   plot.legend("option: y1=0 linetype 2", 9);
   plot.plotting();
}
```

Output

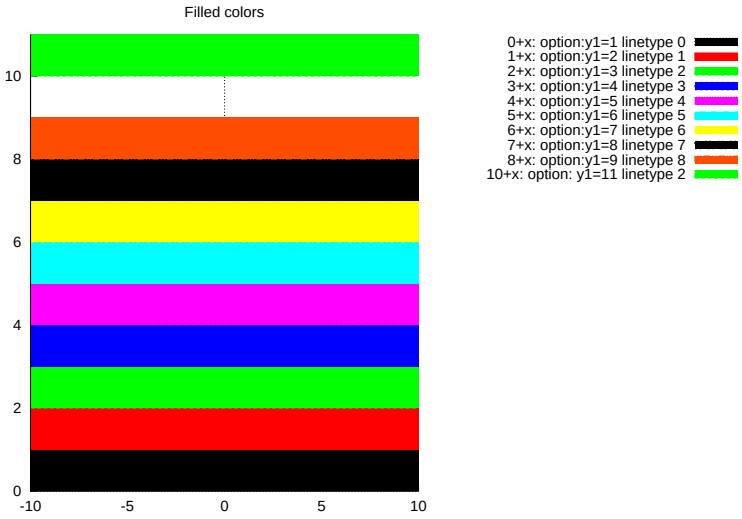


Example 13
This example illustrates the plot type **PLOT_PLOTTYPE_FILLEDCURVES**.

```
/* File: filledcolors2.ch */
#include<math.h>
#include<chplot.h>

#define N 100
```

```
double func(double x, void *param) {
   double offset;
   double y;

   offset = *(double*)param;
   y = offset;
   return y;
}


int main() {
   CPlot plot;
   double offset, x0, xf;
   int i;
   char option[64], legend[64];

   x0 = -10;
   xf = 10;

   plot.title("Filled colors");
   plot.legendOption("outside");
   plot.label(PLOT_AXIS_X, "");
   plot.label(PLOT_AXIS_Y, "");
   plot.axisRange(PLOT_AXIS_X, -10, 10);
   plot.axisRange(PLOT_AXIS_Y, 0, 11);

   for(i=0; i<9; i++) {
     offset = i;
     sprintf(option, "y1=%d linetype %d", i+1, i);
     sprintf(legend, "%d+x: option:%s", i, option);
     plot.funcp2D(x0, xf, N, func, &offset);
     plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, i, option);
     plot.legend(legend, i);
   }
   offset = 10;
   plot.funcp2D(x0, xf, N, func, &offset);
   plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, 9, "y1=11 linetype 2");
   plot.legend("10+x: option: y1=11 linetype 2", 9);
   plot.plotting();

}
```

Output

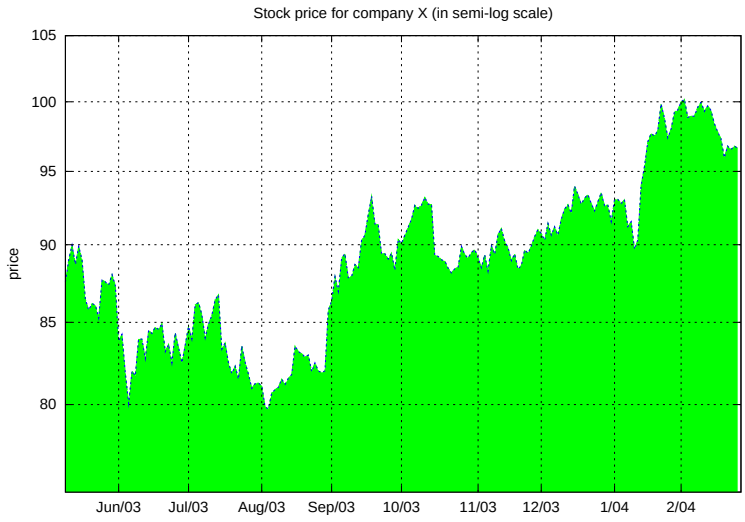


Example 14
This example illustrates the plot type **PLOT_PLOTTYPE_FILLEDCURVES**. This plot for stock prices uses the same data file in Example 10.

```
/* File: stockprice.ch */
/* File: finance.ch to process data in finance.dat

  data and indicators in finance.dat
  # data file layout:
  # date, open, high, low, close, volume,
  # 50-day moving average volume, Intraday Intensity,
  # 20-day moving average close,
  # upper Bollinger Band, lower Bollinger Band
*/

#include <chplot.h>

int main() {
    class CPlot plot;
    int linetype, linewidth;

    plot.title("Stock price for company X (in semi-log scale)");
    plot.label(PLOT_AXIS_X, "");
    plot.label(PLOT_AXIS_Y, "price");
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot.dataFile("finance.dat", "using 0:5");
    plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, 0, "y1=0 linetype 2");
    plot.dataFile("finance.dat", "using 0:5");
    linetype = 3;
    linewidth = 2;
    plot.plotType(PLOT_PLOTTYPE_LINES, 1);
    plot.lineType(1, line_type, line_width);
    plot.axisRange(PLOT_AXIS_X, 50, 253);
    plot.axisRange(PLOT_AXIS_Y, 75, 105);
    plot.scaleType(PLOT_AXIS_Y, PLOT_SCALETYPE_LOG);
    plot.grid(PLOT_ON, "front linewidth 2");
    plot.ticsPosition(PLOT_AXIS_Y, 105, 100, 95, 90, 85, 80);
    /* use the code below for X-label for C++ or Ch */
    /*
```

```
      plot.ticsPosition(PLOT_AXIS_Y, 105);
      plot.ticsPosition(PLOT_AXIS_Y, 100);
      plot.ticsPosition(PLOT_AXIS_Y, 95);
      plot.ticsPosition(PLOT_AXIS_Y, 90);
      plot.ticsPosition(PLOT_AXIS_Y, 85);
      plot.ticsPosition(PLOT_AXIS_Y, 80);
    */
    plot.ticsLabel(PLOT_AXIS_X, "Jun/03", 66, "Jul/03", 87, "Aug/03", 109, "Sep/03", 130,
                   "10/03", 151, "11/03", 174, "12/03", 193, "1/04", 215, "2/04", 235);
;
    /* use the code below for X-label for C++ or Ch */
    /*
      plot.ticsLabel(PLOT_AXIS_X, "6/03", 66);
      plot.ticsLabel(PLOT_AXIS_X, "7/03", 87);
      plot.ticsLabel(PLOT_AXIS_X, "8/03", 109);
      plot.ticsLabel(PLOT_AXIS_X, "9/03", 130);
      plot.ticsLabel(PLOT_AXIS_X, "10/03", 151);
      plot.ticsLabel(PLOT_AXIS_X, "11/03", 174);
      plot.ticsLabel(PLOT_AXIS_X, "12/03", 193);
      plot.ticsLabel(PLOT_AXIS_X, "1/04", 215);
      plot.ticsLabel(PLOT_AXIS_X, "2/04", 235);
    */
    plot.plotting();
}
```

Output



Example 15

The example illustrates the plot type **PLOT_PLOTTYPE_BOXES** can be found on page 30.

See Also

CPlot::**arrow**(), **CPlot**::**lineType**(), **CPlot**::**pointType**().

CPlot::plotting

Synopsis
#include <**chplot.h**>
void plotting();

Purpose
Generate a plot file or display a plot.

Return Value
None.

Parameters
None.

Description
The plot is displayed or a file is generated containing the plot when this function is called. It shall be called after all the desired plot options are set.

Example
See **CPlot**::**data2D**().

CPlot::point

Synopsis
#include <**chplot.h**>
int point(**double** *x*, **double** *y*, **double** *z*);
Purpose
Add a point to a 2D plot.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x The x coordinate of the point.

y The y coordinate of the point.

z The z coordinate of the point.

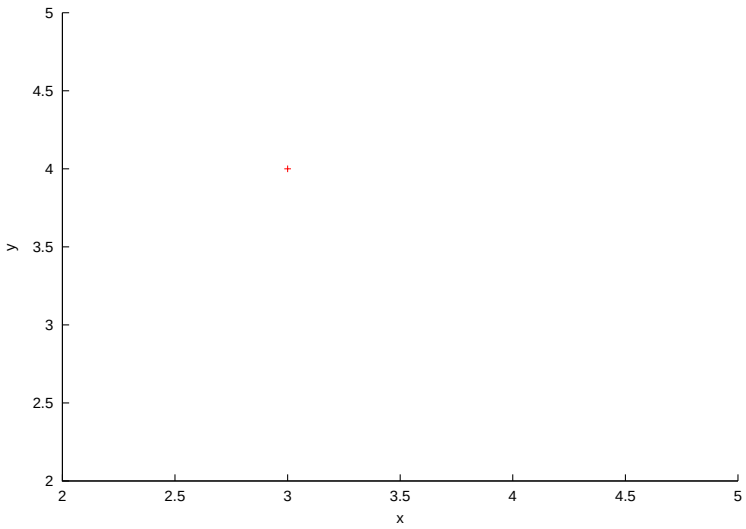
Description
This function adds a point to a plot. It is a convenience function for creation of geometric primitives. A point added with this function counts as a data set for later calls to **CPlot**::**legend**() and **CPlot**::**plotType**(). For 2D rectangular and 3D cartesian plots, *x*, *y*, and *z* are the coordinates of the point specified in units of

the x, y and z axes. However, for 2D plots, *z* is ignored. For 2D polar and 3D cylindrical plots, the point is specified in polar coordinates where *x* is θ, *y* is r, and *z* is unchanged. Again, for 2D plots, *z* is ignored. For 3D plots with spherical coordinates *x* is θ, *y* is ϕ and *z* is r. For 3D plots with points, hidden line removal should be disabled (see **CPlot**::**removeHiddenLine**()) after all data are added.

Example 1

```
/* File: point.ch */
#include <chplot.h>

int main() {
    double x = 3, y = 4;
    class CPlot plot;

    plot.axisRange(PLOT_AXIS_XY, 2, 5); /* one point cannot do autorange */
    plot.point(x, y, 0);
    plot.plotting();
}
```

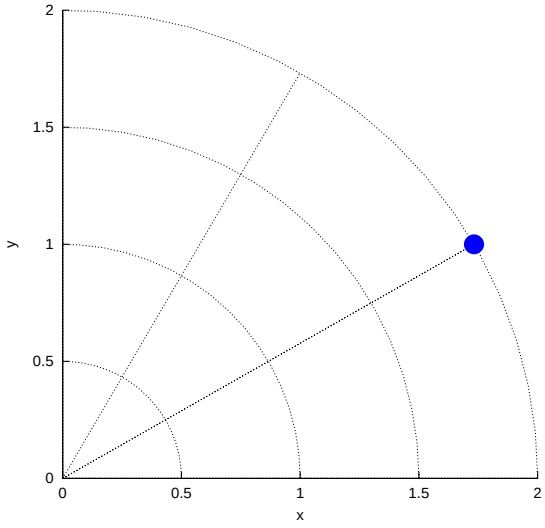
Output



Example 2

```
/* File: point_2.ch */
#include <chplot.h>

int main() {
    double theta1 = 30, r1 = 2;
    class CPlot plot;
    int point_type = 7, point_size = 3;
    char *point_color = "blue";

    plot.grid(PLOT_ON);
    plot.polarPlot(PLOT_ANGLE_DEG);
    plot.point(theta1, r1, 0);
    plot.plotType(PLOT_PLOTTYPE_POINTS, 0);
    plot.pointType(0, point_type, point_size, point_color);
    plot.axisRange(PLOT_AXIS_XY, 0, 2);
    plot.sizeRatio(-1);
    plot.plotting();
}
```

Output



See Also
CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**circle**(), **CPlot**::**line**(), **CPlot**::**outputType**(), **CPlot**::**plotType**(), **CPlot**::**polygon**(), **CPlot**::**rectangle**().

CPlot::pointType

Synopsis
#include <**chplot.h**>
void pointType(**int** *num*, **int** *point_type*, **int** *point_size*], ... /* [**char** *point_color*] */);

Syntax
pointType(*num*, *point_type*, *point_size*)
pointType(*num*, *point_type*, *point_size*, *point_color*)

Purpose
Set the point type, size, and color for points, line-points, etc.

Return Value
None.

Parameters

num The data set to which the point type, size, and color apply.

point_type An integer index representing the desired point type.

point_size A scaling factor for the size of the point used. The point size is *point_size* multiplied by the default size.

point_color color for the point.

Description

Set the desired point type, size, and color for a previously added data set.

Numbering of the data sets starts with zero. The point style and/or marker type for the plot are selected automatically. The default point tye, size, and color can be changed by this member function.

The *point_type* specifies an index for the point type used for drawing the point. The point type varies depending on the terminal type used (see **CPlot**::**outputType**). The value *point_type* is used to change the appearance (color and/or marker type) of a point. It is specified with an integer representing the index of the desired point type. All terminals support at least six different point types. *point_size* is an optional argument used to change the size of the point. The point size is *point_size* multiplied by the default size. If *point_type* and *point_size* are set to zero or a negative number, a default value is used.

An optional fourth argument can specify the color of a point by a color name or RGB value, such as `"blue"` or `"#0000ff"` for color blue. The default point type, size, and color can be changed by the function call

```
plot.pointType(num, pointtype, pointsize, "blue");
```

The color of the point is specified as blue in this example. The valid color names and their corresponding GRB values are listed below.

```
  Color Name         Hexadecimal R   G    B
                                   values
-------------------------------------------
  white               #ffffff = 255 255 255
  black               #000000 =   0   0   0
  gray0               #000000 =   0   0   0
  grey0               #000000 =   0   0   0
  gray10              #1a1a1a =  26  26  26
  grey10              #1a1a1a =  26  26  26
  gray20              #333333 =  51  51  51
  grey20              #333333 =  51  51  51
  gray30              #4d4d4d =  77  77  77
  grey30              #4d4d4d =  77  77  77
  gray40              #666666 = 102 102 102
  grey40              #666666 = 102 102 102
  gray50              #7f7f7f = 127 127 127
  grey50              #7f7f7f = 127 127 127
  gray60              #999999 = 153 153 153
  grey60              #999999 = 153 153 153
```

```
gray70              #b3b3b3 = 179 179 179
grey70              #b3b3b3 = 179 179 179
gray80              #cccccc = 204 204 204
grey80              #cccccc = 204 204 204
gray90              #e5e5e5 = 229 229 229
grey90              #e5e5e5 = 229 229 229
gray100             #ffffff = 255 255 255
grey100             #ffffff = 255 255 255
gray                #bebebe = 190 190 190
grey                #bebebe = 190 190 190
light-gray          #d3d3d3 = 211 211 211
light-grey          #d3d3d3 = 211 211 211
dark-gray           #a9a9a9 = 169 169 169
dark-grey           #a9a9a9 = 169 169 169
red                 #ff0000 = 255   0   0
light-red           #f03232 = 240  50  50
dark-red            #8b0000 = 139   0   0
yellow              #ffff00 = 255 255   0
light-yellow        #ffffe0 = 255 255 224
dark-yellow         #c8c800 = 200 200   0
green               #00ff00 =   0 255   0
light-green         #90ee90 = 144 238 144
dark-green          #006400 =   0 100   0
spring-green        #00ff7f =   0 255 127
forest-green        #228b22 =  34 139  34
sea-green           #2e8b57 =  46 139  87
blue                #0000ff =   0   0 255
light-blue          #add8e6 = 173 216 230
dark-blue           #00008b =   0   0 139
midnight-blue       #191970 =  25  25 112
navy                #000080 =   0   0 128
medium-blue         #0000cd =   0   0 205
royalblue           #4169e1 =  65 105 225
skyblue             #87ceeb = 135 206 235
cyan                #00ffff =   0 255 255
light-cyan          #e0ffff = 224 255 255
dark-cyan           #008b8b =   0 139 139
magenta             #ff00ff = 255   0 255
light-magenta       #f055f0 = 240  85 240
dark-magenta        #8b008b = 139   0 139
turquoise           #40e0d0 =  64 224 208
light-turquoise     #afeeee = 175 238 238
```

```
dark-turquoise     #00ced1 =   0 206 209
pink               #ffc0cb = 255 192 203
light-pink         #ffb6c1 = 255 182 193
dark-pink          #ff1493 = 255  20 147
coral              #ff7f50 = 255 127  80
light-coral        #f08080 = 240 128 128
orange-red         #ff4500 = 255  69   0
salmon             #fa8072 = 250 128 114
light-salmon       #ffa07a = 255 160 122
dark-salmon        #e9967a = 233 150 122
aquamarine         #7fffd4 = 127 255 212
khaki              #f0e68c = 240 230 140
dark-khaki         #bdb76b = 189 183 107
goldenrod          #daa520 = 218 165  32
light-goldenrod    #eedd82 = 238 221 130
dark-goldenrod     #b8860b = 184 134  11
gold               #ffd700 = 255 215   0
beige              #f5f5dc = 245 245 220
brown              #a52a2a = 165  42  42
orange             #ffa500 = 255 165   0
dark-orange        #ff8c00 = 255 140   0
violet             #ee82ee = 238 130 238
dark-violet        #9400d3 = 148   0 211
plum               #dda0dd = 221 160 221
purple             #a020f0 = 160  32 240
```

Examples

See programs and their generated figures for **CPlot**::**point**() and **CPlot**::**plotType**().

See Also
CPlot::**lineType**(), **CPlot**::**point**().
CPlot::**plotType**().

CPlot::polarPlot

Synopsis
#include <**chplot.h**>
void polarPlot(**int** *angle_unit*);

Purpose
Set a 2D plot to use polar coordinates.

Return Value
None.

Parameter
angle_unit Specify the unit for mesurement of an angular position. The following options are available:

PLOT_ANGLE_DEG Angles measured in degree.

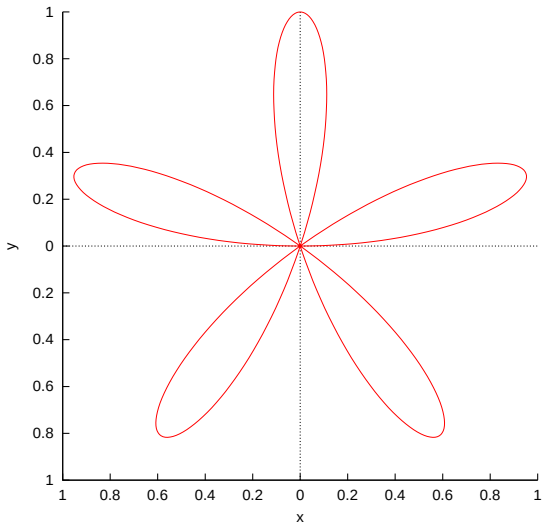
PLOT_ANGLE_RAD Angles measured in radian.

Description
Set a 2D plot to polar coordinates. In polar mode, two numbers, θ and r, are required for each point. First two arguments in member functions **CPlot**::**data2D**() and **CPlot**::**data2DCurve**() are the phase angle θ and magnitude r of points to be plotted.

Example 1
Compare with the example output in **CPlot**::**border**().

```
/* File: polarPlot.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 360;
    array double theta[numpoints], r[numpoints];
    class CPlot plot;

    lindata(0, M_PI, theta);
    r = sin(5*theta);                   // Y-axis data.
    plot.polarPlot(PLOT_ANGLE_RAD);
    plot.data2D(theta, r);
    plot.sizeRatio(1);
    plot.plotting();
}
```

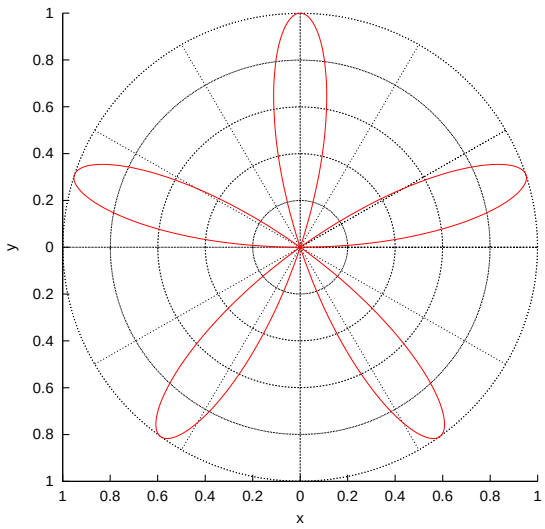
Output



Example 2

```
/* File: polarPlot_2.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 360;
    array double theta[numpoints], r[numpoints];
    class CPlot plot;

    lindata(0, M_PI, theta);
    r = sin(5*theta);                   // Y-axis data.
    plot.polarPlot(PLOT_ANGLE_RAD);
    plot.data2D(theta, r);
    plot.sizeRatio(1);
    plot.grid(PLOT_ON);
    plot.plotting();
}
```

Output



See Also
CPlot::**grid**().

CPlot::polygon

Synopsis
#include <**chplot.h**>
int polygon(**double** *x*[:], **double** *y*[:], **double** *z*[:], ... /* [**int** *num*] */);
Syntax
polygon(*x*, *y*, *z*)
polygon(*x*, *y*, *z*, *num*)

Purpose
Add a polygon to a plot.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x An array containing the x coordinates of the vertices of the polygon.

y An array containing the y coordinates of the vertices of the polygon.

z An array containing the z coordinates of the vertices of the polygon.

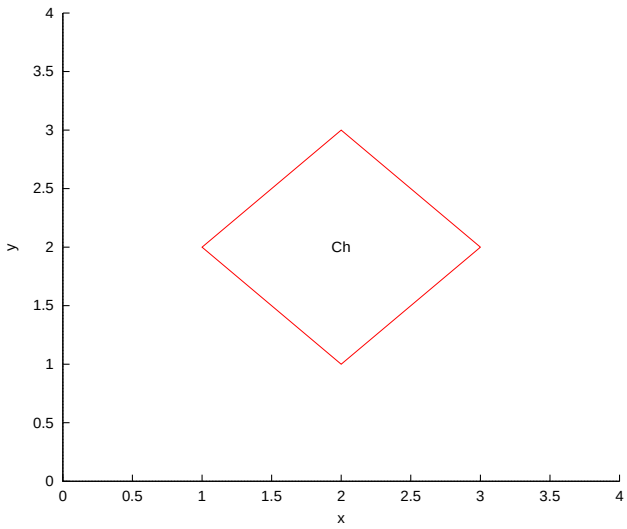
num The number of elements of arrays *x*, *y*, and *z*.

Description
This function adds a polygon to a plot. It is a convenience function for creation of geometric primitives. A polygon added with this function is counted as a data set for later calls to **CPlot**::**legend**() and **CPlot**::**plotType**(). For 2D rectangular plots and 3D cartesian plots, *x*, *y*, and *z* contain the polygon vertices specified in units of the x, y, and z axes. However, for 2D plots, *z* is ignored. For 2D polar and 3D cylindrical plots, the locations of the vertices are specified in polar coordinates where *x* is θ, *y* is r, and *z* is unchanged. Again, for 2D plots, *z* is ignored. For 3D plots with spherical coordinates *x* is θ, *y* is ϕ and *z* is r. Each of the points is connected to the next in a closed chain. The line type and width vary depending on the terminal type used (see **CPlot**::**outputType**). Typically, changing the line type will change the color of the line or make it dashed or dotted. All terminals support at least six different line types.

Example 1

```
/* File: polygon.ch */
#include <chplot.h>

int main() {
    double x[5] = {3, 2, 1, 2, 3}, y[5] = {2, 3, 2, 1, 2};
    class CPlot plot;

    plot.polygon(x, y, NULL);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_XY, 0, 4);
    plot.text("Ch", PLOT_TEXT_CENTER, 2, 2, 0);
    plot.plotting();
}
```

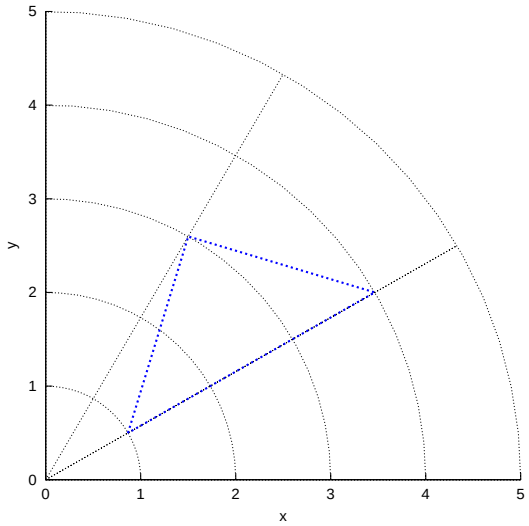
Output



Example 2

```
/* File: polygon_2.ch */
#include <chplot.h>
#include <math.h>

int main(){
    double theta[4] = {30, 60, 30, 30}, r[4] = {1, 3, 4, 1};
    class CPlot plot;

    plot.grid(PLOT_ON);
    plot.polarPlot(PLOT_ANGLE_DEG);
    plot.polygon(theta, r, NULL);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.lineType(0, 3, 4);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_XY, 0, 5);
    plot.plotting();
}
```

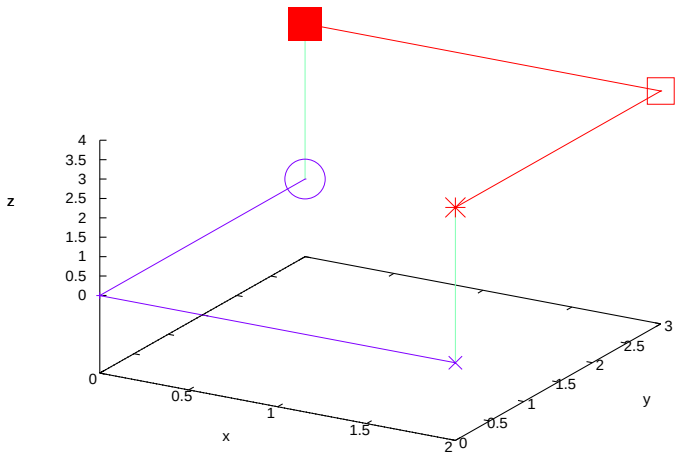
Output



Example 3

```
/* File: polygon_3.ch */
#include <chplot.h>

int main() {
    double x[7] = {0, 2, 2, 2, 0, 0, 0}, y[7] = {0, 0, 0, 3, 3, 3, 0},
           z[7] = {0, 0, 4, 4, 4, 0, 0};
    class CPlot plot;
    int datasetnum, pointtype, pointsize;

    plot.dimension(3);
    plot.polygon(x, y, z);
    plot.point(0, 0, 0);
    plot.point(2, 0, 0);
    plot.point(2, 0, 4);
    plot.point(2, 3, 4);
    plot.point(0, 3, 4);
    plot.point(0, 3, 0);
    for(datasetnum=1, pointtype=1, pointsize=1;
        datasetnum <= 6;
        datasetnum++, pointtype++, pointsize++)
       plot.plotType(PLOT_PLOTTYPE_POINTS,
                     datasetnum, pointtype, pointsize);
    plot.removeHiddenLine(PLOT_OFF);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
}
```

Output



See Also

CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**circle**(), **CPlot**::**line**(), **CPlot**::**outputType**(), **CPlot**::**plotType**(), **CPlot**::**point**(), **CPlot**::**rectangle**().

CPlot::rectangle

Synopsis
#include <chplot.h>
int rectangle(double *x*, **double** *y*, **double** *width*, **double** *height*);
Purpose
Add a rectangle to a 2D plot.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x The x coordinate of the lower-left corner of the rectangle.

y The y coordinate of the lower-left corner of the rectangle.

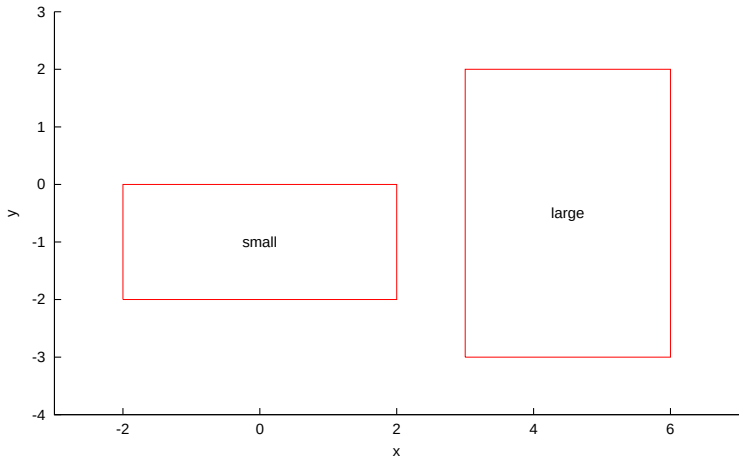
width The width of the rectangle.

height The height of the rectangle.

Description
This function adds a rectangle to a 2D plot. It is a convenience function for creation of geometric primitives. A rectangle added with this function is counted as a data set for later calls to **CPlot**::**legend**() and **CPlot**::**plotType**(). For rectangular plots, *x* and *y* are the coordinates of the lower-left corner of the rectangle. For polar plots, the coordinates of the lower-left corner are given in polar coordinates where *x* is theta and *y* is r. In both cases the *width* and *height* are the dimensions of the rectangle in rectangular coordinates.
Example 1

```
/* File: rectangle.ch */
#include <chplot.h>

int main() {
    double x1 = 3, y1 = -3, width1 = 3, height1 = 5;
    double x2 = -2, y2 = -2, width2 = 4, height2 = 2;
    class CPlot plot;

    plot.rectangle(x1, y1, width1, height1);
    plot.rectangle(x2, y2, width2, height2);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_X, -3, 7);
    plot.axisRange(PLOT_AXIS_Y, -4, 3);
    plot.axis(PLOT_AXIS_XY, PLOT_OFF);
    plot.text("large", PLOT_TEXT_CENTER, 4.5, -0.5, 0);
    plot.text("small", PLOT_TEXT_CENTER, 0, -1, 0);
    plot.plotting();
}
```

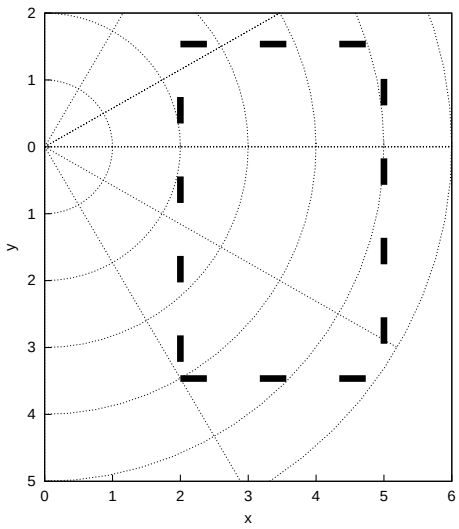
Output



Example 2

```
/* File: rectangle_2.ch */
#include <chplot.h>

int main() {
    double theta1 = -60, r1 = 4, width1 = 3, height1 = 5;
    class CPlot plot;

    plot.grid(PLOT_ON);
    plot.polarPlot(PLOT_ANGLE_DEG);
    plot.rectangle(theta1, r1, width1, height1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.lineType(0, 0, 25);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_X, 0, 6);
    plot.axisRange(PLOT_AXIS_Y, -5, 2);
    plot.axis(PLOT_AXIS_XY, PLOT_OFF);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.plotting();
}
```

Output



See Also
CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**circle**(), **CPlot**::**line**(), **CPlot**::**outputType**(), **CPlot**::**plotType**(), **CPlot**::**point**(), **CPlot**::**polygon**().

CPlot::removeHiddenLine

Synopsis
#include <**chplot.h**>
void removeHiddenLine(**int** *flag*);

Purpose
Enable or disable hidden line removal for 3D surface plots.

Return Value
None.

Parameter
flag This parameter can be set to:

PLOT_ON Enable hidden line removal.

PLOT_OFF Disable hidden line removal.

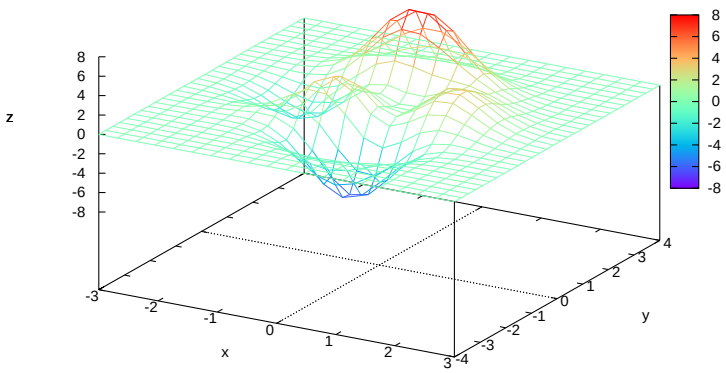
Description
Enable or disable hidden line removal for 3D surface plots. This option is only valid for 3D plots with a plot type set to **PLOT_PLOTTYPE_LINES** or **PLOT_PLOTTYPE_LINESPOINTS** with surface display enabled. By default hidden line removal is enabled. This function should be called after data set are added to the plot. The **PLOT_CONTOUR_SURFACE** option for **CPlot**::**contourMode**() does not work when

hidden line removal is enabled. **CPlot**::**point**() cannot be used when hidden line removal is enabled. By default, the hidden lines are removed.

Example 1
Compare with the output for the example in **CPlot**::**data3D**(), **CPlot**::**contourLabel**(), and **CPlot**::**contourLevels**().

```
/* File: removeHiddenLine.ch */
#include <math.h>
#include <chplot.h>

int main() {
    double x[20], y[30], z[600];
    int i,j;
    class CPlot plot;

    lindata(-3, 3, x);
    lindata(-4, 4, y);
    for(i=0; i<20; i++) {
        for(j=0; j<30; j++) {
            z[30*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.removeHiddenLine(PLOT_OFF);
    plot.plotting();
}
```

Output

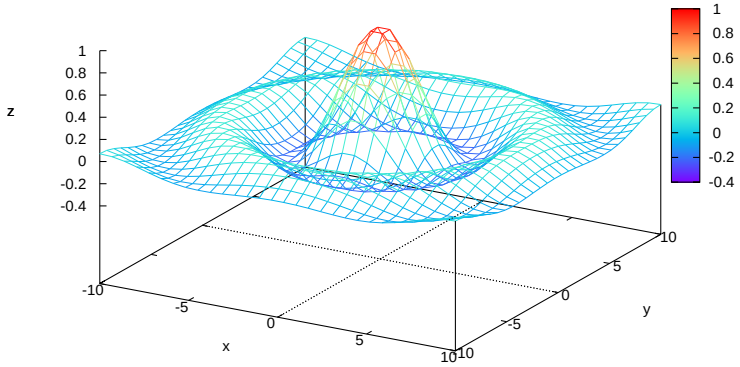


Example 2

```
/* File: removeHiddenLine_2.ch */
#include <math.h>
#include <chplot.h>
```

```
int main() {
    double x[30], y[30], z[900];
    double r;
    int i, j;
    class CPlot plot;

    lindata(-10, 10, x);
    lindata(-10, 10, y);
    for(i=0; i<30; i++) {
        for(j=0; j<30; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[30*i+j] = sin(r)/r;
        }
    }
    plot.data3D(x, y, z);
    plot.removeHiddenLine(PLOT_OFF);
    plot.plotting();
}
```

Output



See Also
CPlot::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**contourMode**(), **CPlot**::**plotType**(), **CPlot**::**point**(), **CPlot**::**showMesh**().

CPlot::scaleType

Synopsis
#include <**chplot.h**>
void scaleType(**int** *axis*, **int** *scale_type*, ... /* [**double** *base*] */);

Syntax
scaleType(*axis*, *scale_type*)
scaleType(*axis*, *scale_type*, *base*)

Purpose
Set the axis scale type for a plot.

Return Value
None.

Parameters
axis The axis to be modified. Valid values are:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

scale_type The scale type for the specified axis. Valid values are:

PLOT_SCALETYPE_LINEAR Use a linear scale for a specified axis.

PLOT_SCALETYPE_LOG Use a logarithmic scale for a specified axis.

base The base for a log scale. For log scales the default base is 10.

Description
Using this function an axis can be modified to have a logarithmic scale. By default, axes are in linear scale. For a semilog scale in the x-coordinate, call the member function

```
plot.scaleType(PLOT_AXIS_X, PLOT_SCALETYPE_LOG);
```

For a logarithmic base 2, call the member function

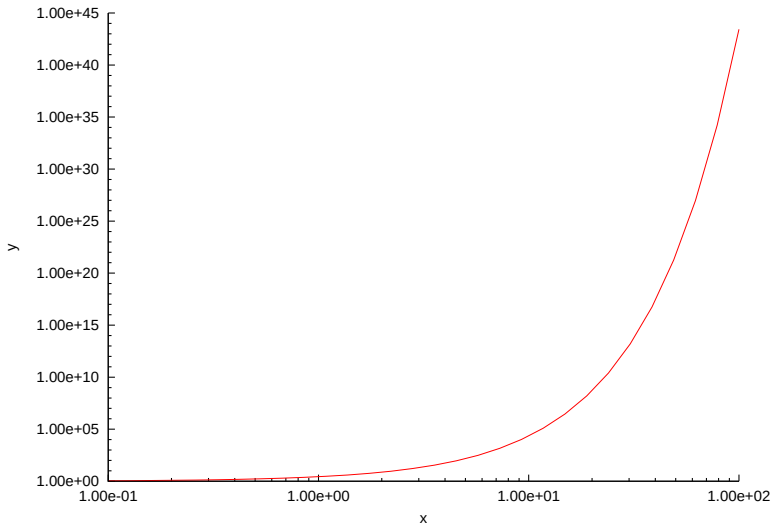
```
plot.scaleType(PLOT_AXIS_X, PLOT_SCALETYPE_LOG, 2); // log base = 2
```

Example

```
/* File: scaleType.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 30;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    logspace(x, -1, 2);
    y = exp(x);                                  // Y-axis data.
    plot.scaleType(PLOT_AXIS_XY, PLOT_SCALETYPE_LOG);
```

```
    plot.ticsFormat(PLOT_AXIS_XY, "%.2e");
    plot.data2D(x, y);
    plot.plotting();
}
```

Output



CPlot::showMesh

Synopsis
#include <chplot.h>
void showMesh(int *flag*);

Purpose
Display 3D data.

Return Value
None.

Parameters
flag This parameter can be set to:

PLOT_ON Enable the display of 3D data.

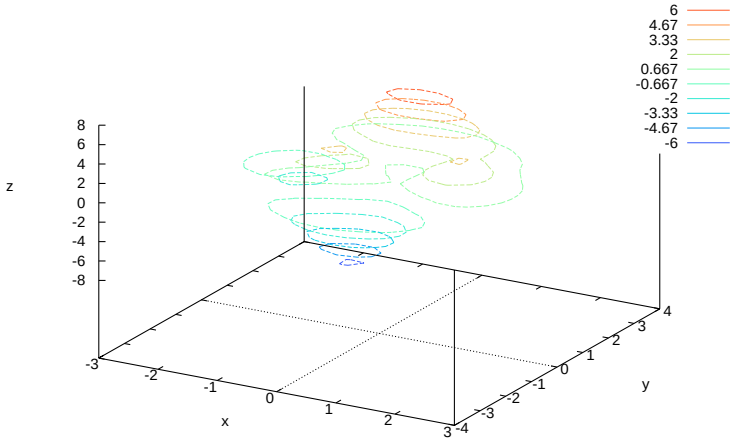
PLOT_OFF Disable the display of 3D data.

Description
Enable or disable the display of 3D data. If this option is disabled, data points or lines will not be drawn. This option is useful with **CPlot::contourMode**() to display contour lines without the display of a surface grid.

Example 1
Compare with the output for the example in **CPlot**::**contourLabel**().

```
/* File: showMesh.ch */
#include <math.h>
#include <chplot.h>

int main() {
    double x[20], y[30], z[600];
    double levels[10];
    int datasetnum =0, i, j;
    class CPlot plot;

    lindata(-6, 6, levels);
    lindata(-3, 3, x);
    lindata(-4, 4, y);
    for(i=0; i<20; i++) {
        for(j=0; j<30; j++) {
            z[30*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.contourLabel(PLOT_ON);
    plot.showMesh(PLOT_OFF);
    plot.contourMode(PLOT_CONTOUR_SURFACE);
    plot.contourLevels(levels);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
}
```

Output

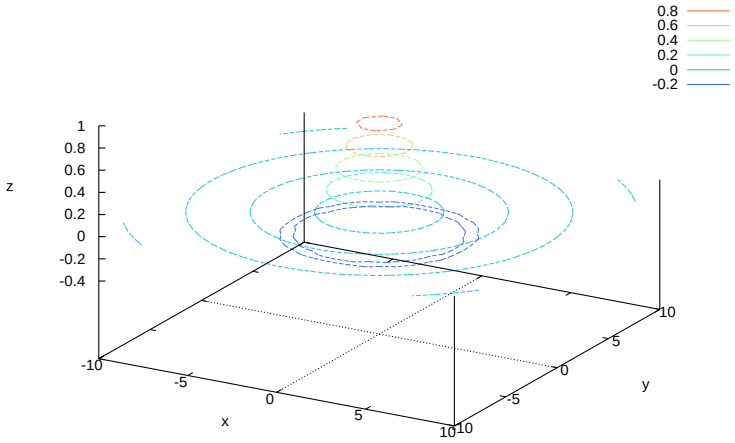


Example 2

```
/* File: showMesh_2.ch */
#include <math.h>
```

```
#include <chplot.h>

int main() {
    double x[30], y[30], z[900];
    double r;
    int datasetnum =0, i, j;
    class CPlot plot;

    lindata(-10, 10, x);
    lindata(-10, 10, y);
    for(i=0; i<30; i++) {
        for(j=0; j<30; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[30*i+j] = sin(r)/r;
        }
    }
    plot.data3D(x, y, z);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.contourLabel(PLOT_ON);
    plot.showMesh(PLOT_OFF);
    plot.contourMode(PLOT_CONTOUR_SURFACE);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
}
```

Output



See Also
CPlot::data3D(), CPlot::data3DCurve(), CPlot::data3DSurface(), CPlot::contourMode(), CPlot::plotType().

CPlot::size

Synopsis
#include <chplot.h>

void size(**double** *x_scale*, **double** *y_scale*);

Purpose
Scale the plot itself relative to the size of the output file or canvas.

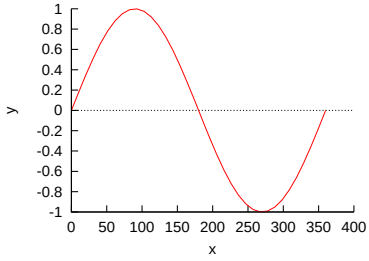
Return Value
None.

Parameters
x_scale A positive multiplicative scaling factor in the range (0, 1) for the x direction.

y_scale A positive multiplicative scaling factor in the range (0, 1) for the y direction.

Description
This function can be used to scale a plot itself relative to the size of the output file or canvas. If the plot is displayed on a screen, the plot is scaled relative to the size of its canvas. If the plot is saved to a file, or output to the stdout stream, the size of the plot image is scaled relative to the output file. For a plot with subplots, this function should be called before **CPlot**::**subplot**() is called.

Example
```
/* File: size.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.data2D(x, y);
    plot.size(.5, .5);
    plot.plotting();
}
```

Output


See Also
CPlot::**size3D**(), **CPlot**::**sizeOutput**(), and **CPlot**::**sizeRatio**().

CPlot::size3D

Synopsis
#include <**chplot.h**>
void size3D(**float** *scale*, **float** *z_scale*);

Purpose
Scale a 3D plot.

Return Value
None.

Parameters
scale A positive multiplicative scaling factor that is applied to the entire plot.

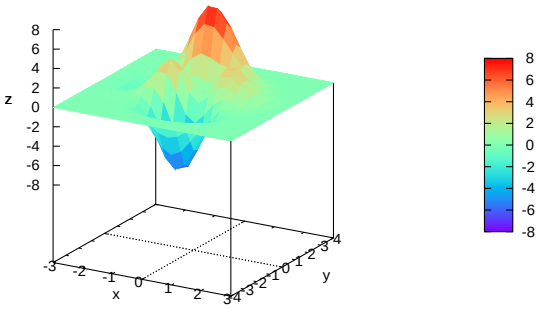
z_scale A positive multiplicative scaling factor that is applied to the z-axis only.

Description
This function can be used to scale a 3D plot to the desired size. By default, *scale* and *z_scale* are both 1.

Example
Compare with the output for examples in **CPlot**::**data3D**() and **CPlot**::**data3DSurface**().

```
/* File: size3D.ch */
#include <math.h>
#include <chplot.h>

int main() {
    double x[20], y[30], z[600];
    int i,j;
    class CPlot plot;

    lindata(-3, 3, x);
    lindata(-4, 4, y);
    for(i=0; i<20; i++) {
        for(j=0; j<30; j++) {
            z[30*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.size3D(0.5, 2);
    plot.plotting();
}
```

Output



See Also
CPlot::**size**().

CPlot::sizeOutput

Synopsis
#include <**chplot.h**>
void sizeOutput(int *xpixels*, **int** *ypixels*);

Purpose
Change the size of an output file.

Return Value
None.

Parameters
xpixels A positive integral number for the number of pixels in the x direction.

ypixels A positive integral number for the number of pixels in the y direction.

Description
This function can be used to change the default size (640x480) of an output image file for the plot.

Example
See Also
CPlot::**size**(), **CPlot**::**size3D**(), and **CPlot**::**sizeRatio**().

CPlot::sizeRatio

Synopsis
#include <**chplot.h**>
void sizeRatio(**float** *ratio*);

Purpose
Change the aspect ratio for a plot.

Return Value
None.

Parameter
ratio The aspect ratio for the plot.

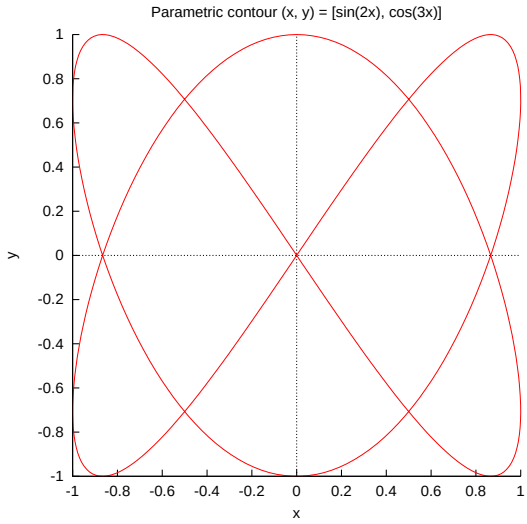
Description
The function sets the aspect ratio for the plot. The meaning of *ratio* changes depending on its value. For a positive *ratio*, it is the ratio of the length of the y-axis to the length of the x-axis. So, if *ratio* is 2, the y-axis will be twice as long as the x-axis. If *ratio* is zero, the default aspect ratio for the terminal type is used. If it is negative, *ratio* is the ratio of the y-axis units to the x-axis units. So, if *ratio* is -2, one unit on the y-axis will be twice as long as one unit on the x-axis.

Portability
The aspect ratio specified is not exact on some terminals. To get a true ratio of 1, for example, some adjustment may be necessary.

Example
Compare with output for example in **plotxy**().

```
/* File: sizeRatio.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 360;
    array double t[numpoints], x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 2*M_PI, t);
    x = sin(2*t);
    y = cos(3*t);
    plot.data2D(x, y);
    plot.title("Parametric contour (x, y) = [sin(2x), cos(3x)]");
    plot.label(PLOT_AXIS_X, "x");
    plot.label(PLOT_AXIS_Y, "y");

    /* both x and y axes the same length */
    plot.sizeRatio(1);
    plot.plotting();
}
```

Output



See Also
CPlot::**size**(), **CPlot**::**size3D**().

CPlot::smooth

Synopsis
#include <**chplot.h**>
void smooth(**int** num, **char** * *option*);

Syntax
smooth(*num*, *option*)

Purpose
Smooth a plotting curve by interpolation or approximation of data.

Return Value
None.

Parameters
num The data set for the curve to be smooth.

option The options smooth a curve.

Description

This function can be used to readily plot a smooth curve through your data points for a 2D plotting by interpolation and approximation of data. However, sophisticated data processing may be performed by preprocessing the data in your Ch program.

The argument `option` of string type with the following values can be used to smooth the data points.

`{unique | frequency | csplines | acsplines | bezier | sbezier}`

The `unique` and `frequency` plot the data after making them monotonic. Each of the other options uses the data to determine the coefficients of a continuous curve between the end points of the data. This curve is then plotted in the same manner as a function, that is, by finding its value at uniform intervals along the abscissa and connecting these points with straight line segments (if a line style is chosen).

If the axis range is determined automatically, the ranges will be computed such that the plotted curve lies within the borders of the graph.

If **CPlot**::**axisRange**() is called, and the smooth option is either `acspline` or `cspline`, the sampling of the generated curve is done across the intersection of the x range covered by the input data and the fixed abscissa range as defined by **CPlot**::**axisRange**() for x-axis.

If too few points are available to allow the selected option to be applied, an error message is produced. The minimum number is one for `unique` and `frequency` four for `acsplines`, and three for the others.

`"acsplines"` The `acsplines` option approximates the data with a "natural smoothing spline". After the data are made monotonic in x a curve is piecewise constructed from segments of cubic polynomials whose coefficients are found by the weighting the data points; the weights are taken from the third column in the data file or data in the memory. If the data is from a data file, that default can be modified by the third entry in the option `using` list for **CPlot**::**dataFile**() as shown below.

```
plot.dataFile("datafile", "using 1:2:(1.0)");
plot.smooth(plot.dataSetNum(), "acsplines");
```

`bezier` The `bezier` option approximates the data with a Bezier curve of degree n (the number of data points) that connects the endpoints.

`"csplines"` The `csplines` option connects consecutive points by natural cubic splines after rendering the data monotonic.

`"sbezier"` The `sbezier` option first renders the data monotonic and then applies the `bezier` algorithm.

`"unique"` The `unique` option makes the data monotonic in x; points with the same x-value are replaced by a single point having the average y-value. The resulting points are then connected by straight line segments.

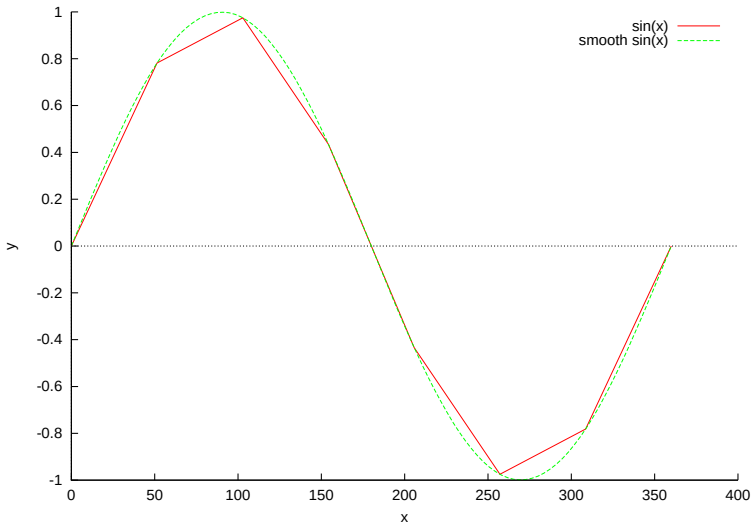
`"frequency"` The `frequency` option makes the data monotonic in x; points with the same x-value are replaced by a single point having the summed y-values. The resulting points are then connected by straight line segments.

Example 1

```
/* File: smooth.ch */
#include <math.h>
#include <chplot.h>
```

```
int main() {
    int numpoints = 8;
    array double x[numpoints], y[numpoints], y2[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    y2 = sin(x*M_PI/180);
    plot.data2D(x, y);
    plot.data2D(x, y2);
    plot.smooth(1, "csplines");
    plot.legend("sin(x)", 0);
    plot.legend("smooth sin(x)", 1);
    plot.plotting();
}
```

Output



Example 2

See an example on page 153 for **CPlot**:**plotType**() on how data for an error bar are smoothed by the option `csplines`.

CPlot::subplot

Synopsis

#include <**chplot.h**>

int subplot(**int** *row*, **int** *col*);

Purpose

Create a group of subplots.

Return Value

This function returns 0 on success and -1 on failure.

Parameters

row The number of rows in the subplot.

col The number of columns in the subplot.

Description

This function allocates memory for (*row***col*)-1 instances of the **CPlot** class to be used in a subplot. The location of the drawing origin and the scale for each element of the subplot is set automatically. The first element of the subplot (an instance of the **CPlot** class) is created normally by the user before this function is called. The remaining elements of the subplot are created and stored internally when this function is called. These elements are accessible through a pointer returned by the **CPlot**::**getSubplot**() function. Calling **CPlot**::**subplot**() with *row* = *col* = 1 has no effect.

Example

See **CPlot**::**getSubplot**().

See Also

CPlot::**origin**(),**CPlot**::**getSubplot**(), **CPlot**::**size**().

CPlot::text

Synopsis

#include <**chplot.h**>
void text(**string_t** string, **int** *just*, **double** *x*, **double** *y*, **double** *z*);

Purpose

Add a text string to a plot.

Return Value

None.

Parameters

string The string to be added at location (*x*,*y*) for 2D plots or (*x*,*y*,*z*) for 3D plots. The location of the text is in plot coordinates.

just The justification of the text. Valid values are:

PLOT_TEXT_LEFT The specified location is the left side of the text string.

PLOT_TEXT_RIGHT The specified location is the right side of the text string.

PLOT_TEXT_CENTER The specified location is the center of the text string.

x The x position of the text.

y The y position of the text.

z The z position of the text. This argument is ignored for a 2D plot.

Description
This function can be used to add text to a plot at an arbitrary location.

Example
See **CPlot**::**arrow**(), **CPlot**::**polygon**(), and **CPlot**::**rectangle**().

CPlot::tics

Synopsis
#include <**chplot.h**>
void tics(**int** *axis*, **int** *flag*)

Purpose
Enable or disable display of axis tics.

Return Value
None.

Parameters
axis The axis which labels are added to. This parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

flag This parameter can be set to:

PLOT_ON Enable drawing of tics for the specified axis.

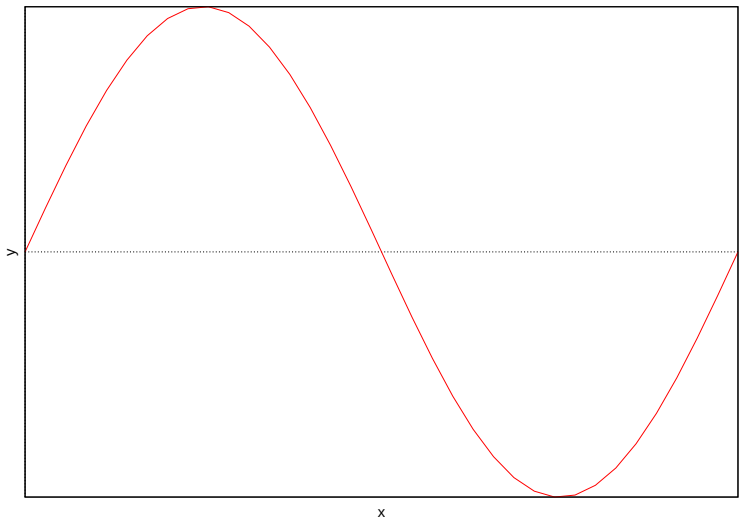
PLOT_OFF Disable drawing of tics for the specified axis.

Description
Enable or disable the display of tics and numerical labels for an axis. By default, tics are displayed for the x and y axes on 2D plots and for the x, y and z axes on 3D plots.

Example
Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: tics.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.data2D(x, y);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.tics(PLOT_AXIS_XY, PLOT_OFF);
    plot.plotting();
}
```

Output



See Also
CPlot::**ticsDirection**(), **CPlot**::**ticsFormat**(), **CPlot**::**ticsLabel**(), **CPlot**::**ticsLevel**(), **CPlot**::**ticsLocation**(), and **CPlot**::**ticsMirror**().

CPlot::ticsDay

Synopsis
#include <**chplot.h**>
void ticsDay(**int** *axis*);

Purpose
Set axis tic-marks to days of the week.

Return Value

None.

Parameter

axis The *axis* parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

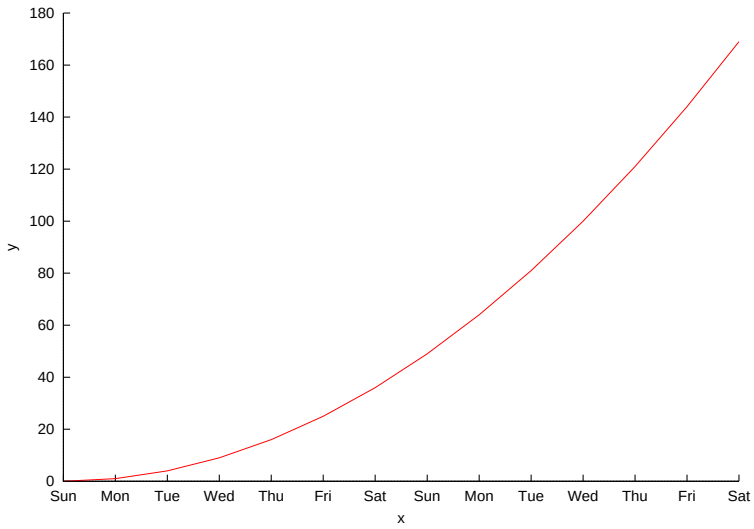
Description

Sets axis tic marks to days of the week (0=Sunday, 6=Saturday). Values greater than 6 are converted into the value of modulo 7.

Example

```
/* File: ticsDay.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 14;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, numpoints-1, x);
    y = x.*x;
    plot.ticsDay(PLOT_ON);
    plot.data2D(x, y);
    plot.plotting();
}
```

Output



See Also
CPlot::**ticsMonth**(), **CPlot**::**ticsLabel**().

CPlot::ticsDirection

Synopsis
#include <**chplot.h**>
void ticsDirection(**int** *direction*);

Purpose
Set the direction in which the tic-marks are drawn for an axis.

Return Value
None.

Parameter
direction Direction tic-marks are drawn. Can be set to:

PLOT_TICS_IN Draw axis tic-marks inward.

PLOT_TICS_OUT Draw axis tic-marks outward.

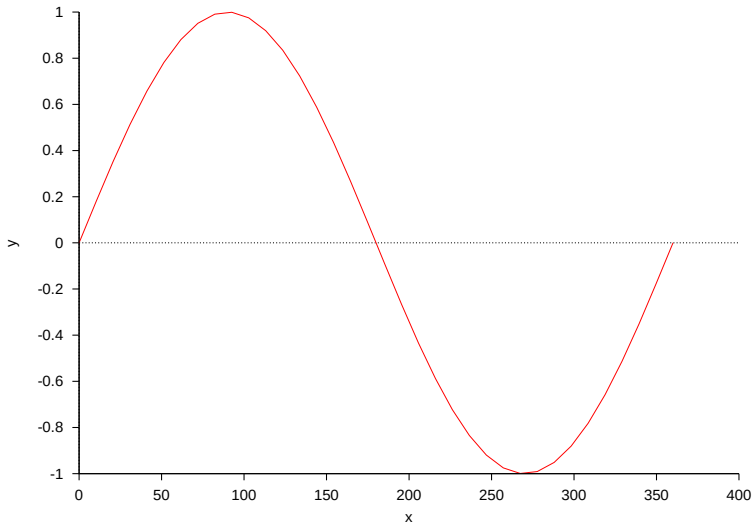
Description
Set the direction in which tic-marks are drawn in the inward or outward direction from the axes. The default is **PLOT_TICS_IN**.

Example
Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: ticsDirection.ch */
#include <math.h>
```

```
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.ticsDirection(PLOT_TICS_OUT);
    plot.data2D(x, y);
    plot.plotting();
}
```

Output



See Also
CPlot::**ticsDay**(), **CPlot**::**ticsLabel**(), **CPlot**::**ticsLocation**(), and **CPlot**::**ticsMonth**().

CPlot::ticsFormat

Synopsis
#include <chplot.h>
void ticsFormat(int *axis*, **string_t** *format*);

Purpose
Set the number format for tic labels.

Return Value
None.

Parameters
axis The axis to be modified. Valid values are:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

format A C-style conversion specifier. Any conversion specifier suitable for double precision floats is acceptable, but other formats are available.

Description

This function allows for custom number formats for tic-mark labels. The default format is `"%g"`. The table below gives some of the available tics formats in C style.

Format	Effect
`%f`	Decimal notation for a floating point number. By default, 6 digits are displayed after the decimal point.
`%e` or `%E`	Scientific notation for a floating point value. There is alwayse one digit to the left of the decimal point. By default, 6 digits are displayed to the right of the decimal point.
`%g` or `%G`	A floating point number. If the value requires an exponent less than -4 or greater than the precision, e or E is used, otherwise f is used.

These format specifiers can be used with the standard C flag characters (-, +, #, etc.), minimum field width specifications and precision specifiers for function **fprintf**(). See **fprintf**() for details.

Examples:

format	number	output
”%f”	234.5678	234.567800
”%.2f”	234.5678	234.57
”%e”	123.456	0.123456e+03
”%E”	123.456	0.123456E+03
”%5.0f”	125.0	125
”%#5.0f”	125.0	125.

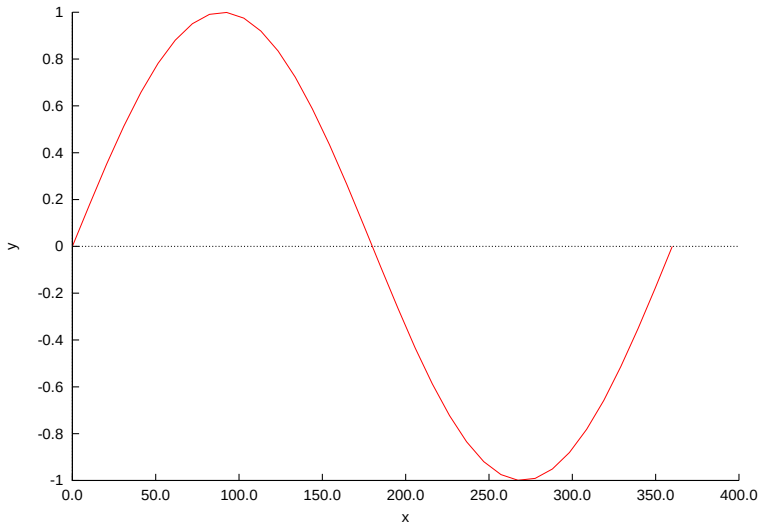
Example

Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: ticsFormat.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
```

```
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.data2D(x, y);
    plot.ticsFormat(PLOT_AXIS_X, "%.1f");
    plot.plotting();
}
```

Output



See Also
CPlot::**ticsLabel**(), **fprintf**().

CPlot::ticsLabel

Synopsis
#include <**chplot.h**>
void ticsLabel(**int** *axis*, ... /* [[**string_t** *label*, **double** *position*], ...] */);

Syntax
ticsLabel(*axis*) /* does nothing */
ticsLabel(*axis*, *label*, *position*)
ticsLabel(*axis*, *label*, *position*, *label2*, *position2*)
etc.

Purpose
Add tic-marks with arbitrary labels to an axis.

Return Value
None.

Parameters

axis The axis which labels are added to. This parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

label The tic-mark label.

position The position of the tic-mark on the axis.

Description

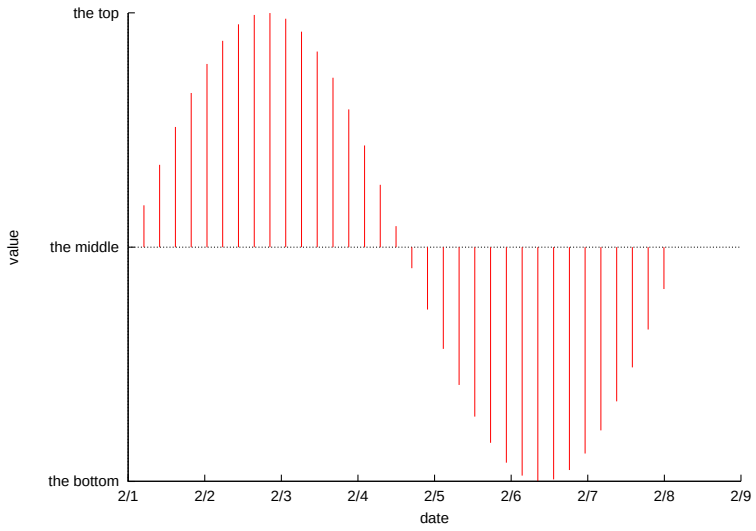
Add tic marks with arbitrary labels to an axis. The axis specification is followed by one or more pairs of arguments. Each pair consists of a label string and a double precision floating point position. This function disables numerical labels for the specified axis. This function can be called multiple times to set tic labels for an axis.

Example

Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: ticsLabel.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.label(PLOT_AXIS_X, "date");
    plot.label(PLOT_AXIS_Y, "value");
    plot.data2D(x, y);
    plot.plotType(PLOT_PLOTTYPE_IMPULSES, 0);
    plot.axisRange(PLOT_AXIS_X, 0, 400);
    plot.ticsLabel(PLOT_AXIS_X, "2/1", 0, "2/2", 50,
                   "2/3", 100, "2/4", 150, "2/5", 200, "2/6", 250,
                   "2/7", 300, "2/8", 350, "2/9", 400);
    plot.ticsLabel(PLOT_AXIS_Y, "the bottom", -1, "the middle", 0,
                   "the top", 1);
    plot.plotting();
}
```

Output



See Also
CPlot::**ticsDirection**(), **CPlot**::**ticsFormat**(), **CPlot**::**ticsLevel**(), **CPlot**::**ticsLocation**().

CPlot::ticsLevel

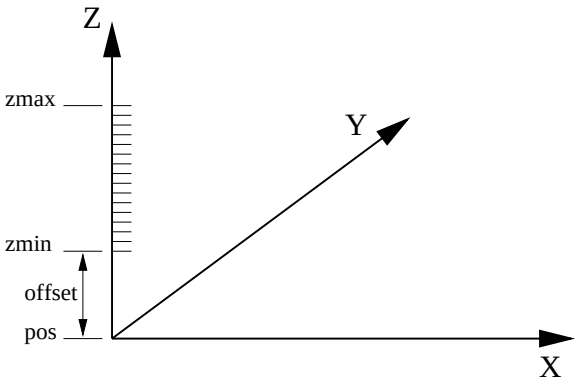
Synopsis
#include <**chplot.h**>
void ticsLevel(**double** *level*);

Purpose
Set the z-axis offset for drawing of tics in 3D plots.

Return Value
None.

Parameters
level The distance between the xy plane and the start of tic-marks on the z axis as a multiple of the full z range. This can be any non-negative number.

Description
This function specifies an offset between the xy plane and the start of z-axis tics-marks as a multiple of the full z range. By default the value for *level* is 0.5, so the z offset is a half of the z axis range. To place the xy-plane at the specified position `pos` on the z-axis, *level* shall equal (zmin-pos)/(zmax-zmin).



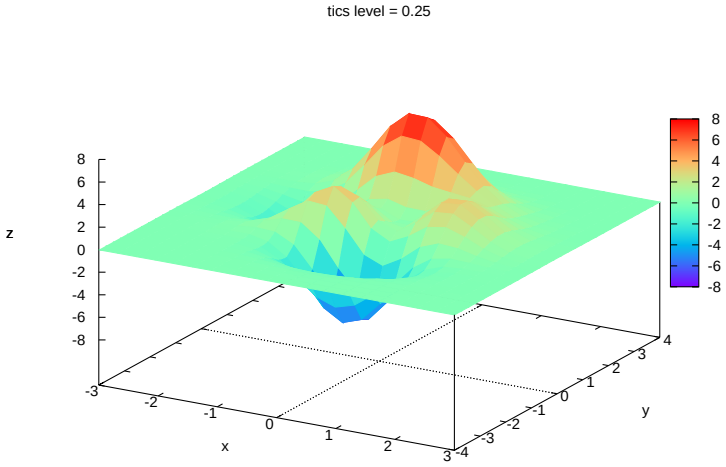
level = (zmin - pos)/(zmax - zmin)

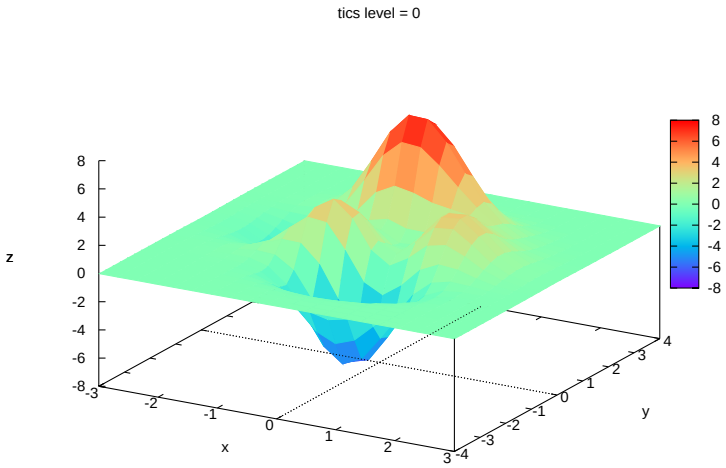
offset = level * (zmax - zmin)

Example
Compare with the output for examples in **CPlot**::**data3D**() and **CPlot**::**data3DSurface**().

```
/* File: ticsLevel.ch */
#include <math.h>
#include <chplot.h>

int main() {
    double x[20], y[30], z[600];
    int i,j;
    class CPlot plot;

    lindata(-3, 3, x);
    lindata(-4, 4, y);
    for(i=0; i<20; i++) {
        for(j=0; j<30; j++) {
            z[30*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3D(x, y, z);
    plot.ticsLevel(.25);
    plot.title("tics level = 0.25");
    plot.plotting();
    plot.ticsLevel(0);
    plot.title("tics level = 0");
    plot.plotting();
}
```

Output





CPlot::ticsLocation

Synopsis

#include <chplot.h>

void ticsLocation(int *axis*, **string_t** *location*)

Purpose

Specify the location of axis tic marks to be on the border or the axis.

Return Value

None.

Parameters

axis The *axis* parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_XY Select the x and y axes.

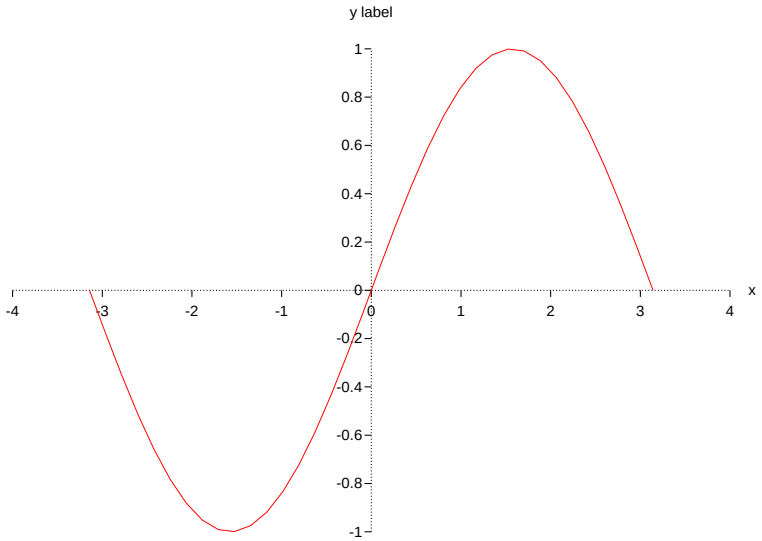
location Tic marks are placed on the plot border with `"border"` or on the axis itself with `"axis"`. By default, tic marks are on the border.

Description
Specify the location of axis tic marks to be on the plot border or the axis itself.

Example
Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: ticsLocation.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(-M_PI, M_PI, x);
    y = sin(x);
    plot.data2D(x, y);
    plot.ticsLocation(PLOT_AXIS_XY, "axis");
    plot.border(PLOT_BORDER_BOTTOM|PLOT_BORDER_LEFT, PLOT_OFF);
    plot.label(PLOT_AXIS_XY, NULL);
    plot.text("y label", PLOT_TEXT_CENTER, 0, 1.15, 0);
    plot.text("x", PLOT_TEXT_CENTER, 4.25, 0, 0);
    plot.margins(-1, -1, 2, -1); /* adjust top margin for y label */
    plot.plotting();
}
```

Output



See Also
CPlot::**tics**(), **CPlot**::**ticsDirection**(), **CPlot**::**ticsFormat**(), **CPlot**::**ticsLabel**, **CPlot**::**ticsLevel**(), **CPlot**::**ticsLocation**, and **CPlot**::**ticsMirror**().

CPlot::ticsMirror

Synopsis
#include <**chplot.h**>
void ticsMirror(**int** *axis*, **int** *flag*)

Purpose
Enable or disable the display of axis tics on the opposite axis.

Return Value
None.

Parameters
axis The axis which labels are added to. This parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

flag This parameter can be set to:

PLOT_ON Enable drawing of tics for the specified axis.

PLOT_OFF Disable drawing of tics for the specified axis.

Description

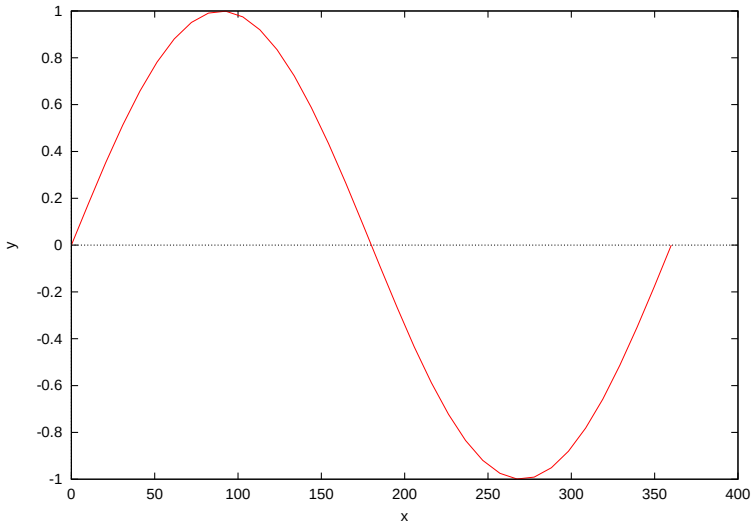
Enable or disable the display of tics on the opposite (mirror) axis. By default, on 2D plots tics on the opposite axis are not displayed. On 3D plots they are displayed

Example

Compare with output for example in **CPlot**::**border**().

```
/* File: ticsMirror.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.data2D(x, y);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot.plotting();
}
```

Output



See Also

CPlot::**tics**(), **CPlot**::**ticsDirection**(), **CPlot**::**ticsFormat**(), **CPlot**::**ticsLabel**(), **CPlot**::**ticsLevel**(), and **CPlot**::**ticsLocation**().

CPlot::ticsMonth

Synopsis
#include <chplot.h>
void ticsMonth(int *axis*);

Purpose
Set axis tic-marks to months.

Return Value
None.

Parameter
axis The axis to be changed. Valid values are:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

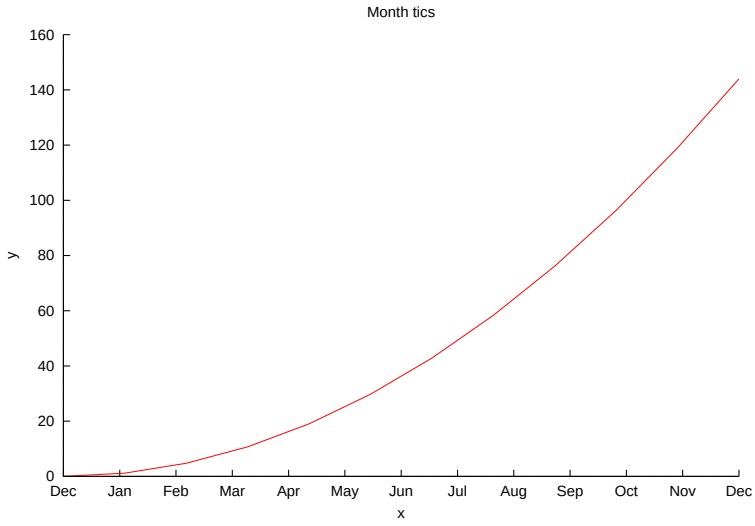
PLOT_AXIS_XYZ Select the x, y, and z axes.

Description
Sets axis tic marks to months of the year (1=January, 12=December). Values greater than 12 are converted into the value of modulo 12.

Example

```
/* File: ticsMonth.ch */
#include <math.h>
#include <chplot.h>

int main() {
    array double x[12], y[12];
    string_t title="Month tics",                        // Define labels.
             xlabel="x",
             ylabel="y";
    class CPlot plot;

    lindata(0, 12, x);
    y = x.*x;
    plot.ticsMonth(PLOT_AXIS_X);
    plot.title(title);
    plot.label(PLOT_AXIS_X, xlabel);
    plot.label(PLOT_AXIS_Y, ylabel);
    plot.data2D(x, y);
    plot.plotting();
}
```

Output



See Also
CPlot::**ticsDay**(), **CPlot**::**ticsLabel**().

CPlot::ticsPosition

Synopsis
#include <**chplot.h**>
void ticsPosition(**int** *axis*, **double** *position1*], ... /* [**double** *position2*], ...] */);

Syntax
ticsPosition(*axis*, *position*)
ticsPosition(*axis*, *position1*, *position2*)
etc.

Purpose
Add tic-marks at the specified positions to an axis.

Return Value
None.

Parameters
axis The axis which tics are added to. This parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

position The position of the tic-mark on the axis.

Description
Add tic marks at the specified positions to an axis. The axis specification is followed by one or more position values of double precision floating point numbers. This function disables numerical labels for the specified axis. This function can be called multiple times to set tic positions for an axis. In this form, the tics do not need to be listed in numerical order.

Examples
See an example on page 153 using for **CPlot**:**ticsPosition**() for the x-axis for date.

See Also
CPlot::**ticsDirection**(), **CPlot**::**ticsFormat**(), **CPlot**::**ticsLevel**(), **CPlot**::**ticsLabel**(), **CPlot**::**ticsLocation**(), **CPlot**::**ticsRange**().

CPlot::ticsRange

Synopsis
#include <**chplot.h**>
void ticsRange(**int** *axis*, **double** *incr*, ... /* [**double** *start*], [**double** *end*] */);

Syntax
ticsRange(*axis*, *incr*) **ticsRange**(*axis*, *incr*, *start*)
ticsRange(*axis*, *incr*, *start*, *end*)

Purpose
Specify the range for a series of tics on an axis.

Return Value
None.

Parameters
axis The *axis* parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

incr The increment between tic marks. By default or when *incr* is 0, the increment between tic marks is calculated internally.

start The starting value for tics.

end The end value for tics.

Description
The range for a series of tics on an axis can be explicitly specified with this function. Any previously specified labeled tic-marks are overridden. The implicit `start`, `incr`, `end` form specifies that a series of tics will be plotted on the axis between the values `start` and `end` with an increment of `incr`. If `end` is not given, it is assumed to be infinity. The increment may be negative. If neither `start` nor `end` is given, `start` is assumed to be negative infinity, `end` is assumed to be positive infinity, and the tics will be drawn at integral multiples of `incr`. If the axis is logarithmic specified by the member function **scaleType**(), the increment will be used as a multiplicative factor.

Example
See **CPlot**::**axisRange**().
See Also
CPlot::**axisRange**(), **CPlot**::**ticsPosition**(), **CPlot**::**ticsLabel**().

CPlot::title

Synopsis
#include <**chplot.h**>
void title(**string_t** *title*);

Purpose
Set the plot title.

Return Value
None.

Parameters
title The plot title.

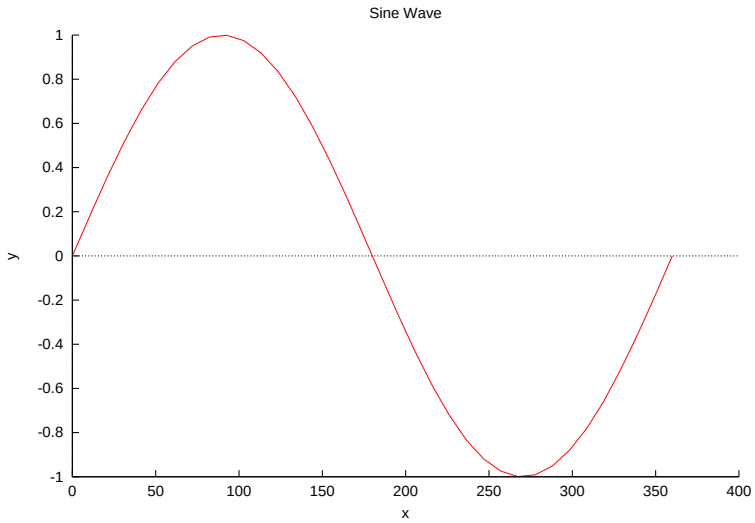
Description
Add a title string to an existing plot variable. For no title, `NULL` can be specified. By default, no title is

specified.

Example
Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
/* File: title.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[numpoints];
    string_t title="Sine Wave";                         // Define labels.
    class CPlot plot;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    plot.title(title);
    plot.data2D(x, y);
    plot.plotting();
}
```

Output



See Also
CPlot::**label**(), **CPlot**::**getLabel**(), **CPlot**::**getTitle**().

fplotxy

Synopsis
#include <**chplot.h**>
int fplotxy(**double** *(*func)(***double** *x),* **double** *x0,* **double** *xf, ...*
/* [**int** *num*, [**string_t** *title*, **string_t** *xlabel*, **string_t** *ylabel*], [**class CPlot** **pl*]] */);

Syntax
fplotxy(*func*, *x0*, *xf*)
fplotxy(*func*, *x0*, *xf*, *num*)
fplotxy(*func*, *x0*, *xf*, *num*, *title*, *xlabel*, *ylabel*)
fplotxy(*func*, *x0*, *xf*, *num*, &*plot*)
fplotxy(*func*, *x0*, *xf*, *num*, *title*, *xlabel*, *ylabel*, &*plot*)

Purpose
Plot a 2D function of *x* in the range $x0 \leq x \leq xf$.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
func A pointer to a function that takes a `double` as an argument and returns a `double`.

x0 The lower bound of the range to be plotted.

xf The upper bound of the range to be plotted.

num The number of points to be plotted. The default is 100.

title The title of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

pl A pointer to an instance of the **CPlot** class.

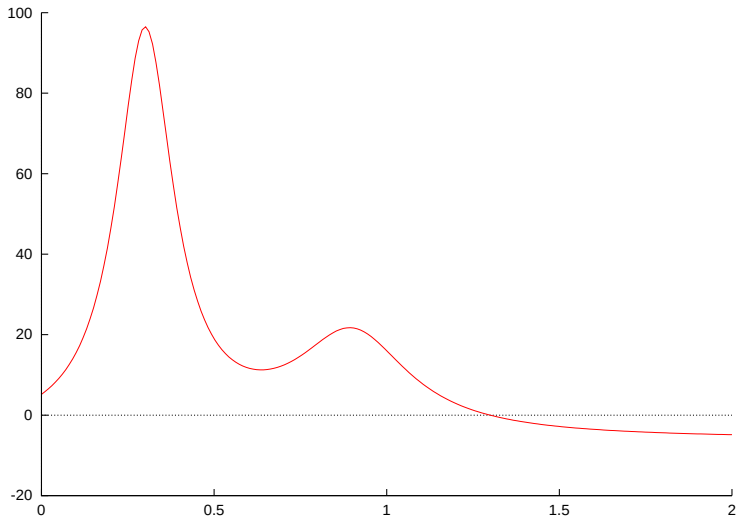
Description
Plot a 2D function of *x* in the range $x0 \leq x \leq xf$. The function to be plotted, *func*, is specified as a pointer to a function that takes a *double* as an argument and returns a *double*. The arguments *x0* and *xf* are the end-points of the range to be plotted. The optional argument *num* specifies how many points in the range are to be plotted. The number of points plotted are evenly spaced in the range. By default, 100 points are plotted. The *title*, *xlabel*, and *ylabel* for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this function. If a non-NULL pointer is passed, it will be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting**() member function. If a previously

initialized *plot* variable is passed, it will be re-initialized with the function parameters. If no pointer or a NULL pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function.

Example

```
/* File: fplotxy.ch */
// Demonstrates the usage of the fplotxy() function.
// Print out a sine wave with appropriate labels and grid range.
#include <math.h>
#include <chplot.h>

int main() {
    double x0 = 0, xf = 2;
    int num = 200;

    /* local function to be plotted */
    double func2(double x) {
        return 1/((x-0.3)*(x-0.3)+0.01) + 1/((x-0.9)*(x-0.9)+0.04) - 6;
    }
    fplotxy(func2,x0,xf,num);
}
```

Output



See Also

CPlot, **fplotxyz**(), **plotxy**(), **plotxyf**(), **plotxyz**(), **plotxyzf**().

fplotxyz

Synopsis
#include <**chplot.h**>
int fplotxyz(**double** *(*func)(***double** *x,* **double** *y),* **double** *x0,* **double** *xf,* **double** *y0,* **double** *yf, ...*
/* [**int** *x_num*, **int** *y_num*, **string_t** *title*, **string_t** *xlabel*, **string_t** *ylabel*, **string_t** *zlabel*],
[**class CPlot** **pl*]]*/);

Syntax
fplotxyz(*func*, *x0*, *xf*, *y0*, *yf*)
fplotxyz(*func*, *x0*, *xf*, *y0*, *yf*, *x_num*, *y_num*)
fplotxyz(*func*, *x0*, *xf*, *y0*, *yf*, *x_num*, *y_num*, &*plot*)
fplotxyz(*func*, *x0*, *xf*, *y0*, *yf*, *x_num*, *y_num*, *title*, *xlabel*, *ylabel*, *zlabel*)
fplotxyz(*func*, *x0*, *xf*, *y0*, *yf*, *x_num*, *y_num*, *title*, *xlabel*, *ylabel*, *zlabel*, &*plot*)

Purpose
Plot a 3D function of *x* and *y* in the range $x0 \leq x \leq xf$ and $y0 \leq y \leq yf$.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
func A pointer to a function that takes two **double** arguments and returns a **double**.

x0 The lower bound of the x range to be plotted.

xf The upper bound of the x range to be plotted.

y0 The lower bound of the y range to be plotted.

yf The upper bound of the y range to be plotted.

x_num The number of points to be plotted. The default is 25.

y_num The number of points to be plotted. The default is 25.

title The title of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

zlabel The z-axis label.

pl A pointer to an instance of the **CPlot** class.

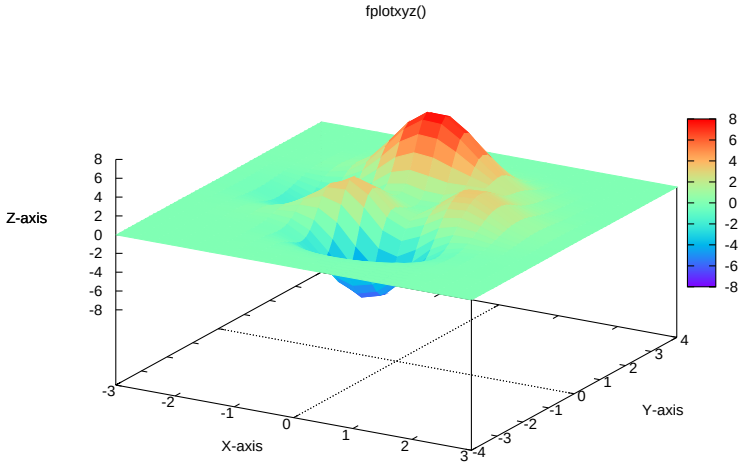
Description

Plot a 3D function of *x* and *y* in the range $x0 \leq x \leq xf$ and $y0 \leq y \leq yf$. The function to be plotted, *func*, is specified as a pointer to a function that takes two **double** arguments and returns a **double**. *x0* and *xf* are the end-points of the *x* range to be plotted. *y0* and *yf* are the end-points of the *y* range to be plotted. The optional arguments *x_num* and *y_num* specify how many points in the *x* and *y* ranges are to be plotted. The number of points plotted are evenly spaced in the ranges. By default, *x_num* and *y_num* are 25. The *title*, *xlabel*, *ylabel*, and *zlabel* for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this function. If a non-NULL pointer is passed, it will be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting**() member function. If a previously initialized **CPlot** variable is passed, it will be re-initialized with the function parameters. If no pointer or a NULL pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function. This function can only be used to plot 3D grid or scatter data, it cannot be used to plot 3D paths.

Example

```
/* File: fplotxyz.ch */
// Demonstrates the usage of the fplotxyz() function.
// Print out a sine wave with appropriate labels and grid range.
#include <math.h>
#include <chplot.h>

int main() {
    string_t title="fplotxyz()",                    // Define labels.
             xlabel="X-axis",
             ylabel="Y-axis",
             zlabel="Z-axis";
    double x0 = -3, xf = 3, y0 = -4, yf = 4;
    int x_num = 20, y_num = 50;

    double func(double x, double y) {               // function to be plotted

        return 3*(1-x)*(1-x)*exp(-(x*x) - (y+1)*(y+1) )
               - 10*(x/5 - x*x*x - pow(y,5))*exp(-x*x-y*y)
               - 1/3*exp(-(x+1)*(x+1) - y*y);
    }
    fplotxyz(func, x0, xf, y0, yf, x_num, y_num, title, xlabel, ylabel, zlabel);
}
```

Output



See Also

CPlot, **fplotxy**(), **plotxy**(), **plotxyf**(), **plotxyz**(), **plotxyzf**().

plotxy

Synopsis
#include <**chplot.h**>
int plotxy(**double** *x*[&], **array double** &*y*, ...
/* [int n] [**string_t** *title*, **string_t** *xlabel*, **string_t** *ylabel*],
[**class CPlot** **pl*] */);

Syntax
plotxy(*x*, *y*)
plotxy(*x*, *y*, *title*, *xlabel*, *ylabel*)
plotxy(*x*, *y*, *title*, *xlabel*, *ylabel*, &*plot*)
plotxy(*x*, *y*, `n`)
plotxy(*x*, *y*, `n`, *title*, *xlabel*, *ylabel*)
plotxy(*x*, *y*, `n`, *title*, *xlabel*, *ylabel*, &*plot*)

Purpose
Plot a 2D data set or initialize an instance of the **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x A one-dimensional array of size n. The value of each element is used for the x-axis of the plot.

y A m x n dimensional array containing m curves, each of which is plotted against x.

n An integer for the number of elements of array `x`.

title The *title* of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

pl A pointer to an instance of the **CPlot** class.

Description
The arrays *x* and *y* can be of any supported data type of real numbers. Conversion of the data to **double** type is performed internally. The optional argument `n` for the number of elements for array `x` is for the compatibility with SIGL C++ graphical library. The *title*, *xlabel*, and *ylabel* for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this function. If a non-NULL pointer is passed, it will be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting**() member function. If a previously initialized **CPlot** variable is passed, it will be re-initialized with the function

parameters. If no pointer or a NULL pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function.

The following code segment

```
class CPlot plot;
plotxy(x, y, "title", "xlabel", "ylabel", &plot);
```

is equivalent to

```
class CPlot plot;
plot.data2D(x, y);
plot.title("title");
plot.label(PLOT_AXIS_X, "xlabel");
plot.label(PLOT_AXIS_Y, "ylabel");
```

The code segment

```
class CPlot plot;
plotxy(x, y, n, "title", "xlabel", "ylabel", &plot);
```

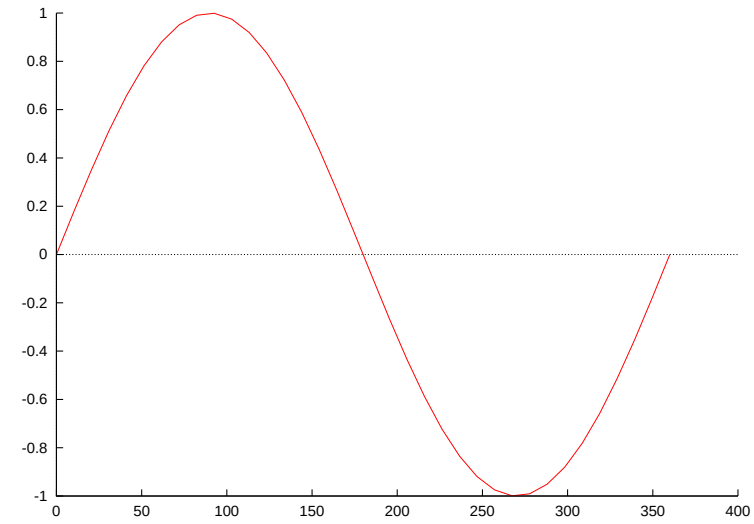
is equivalent to

```
class CPlot plot;
plot.data2DCurve(x, y, n);
plot.title("title");
plot.label(PLOT_AXIS_X, "xlabel");
plot.label(PLOT_AXIS_Y, "ylabel");
```

Example 1

```
/* File: plotxy_5.ch */
#include <math.h>
#include <chplot.h>

int main() {
    array double x[36];

    lindata(0, 360, x);
    plotxy(x, sin(x*M_PI/180));
}
```

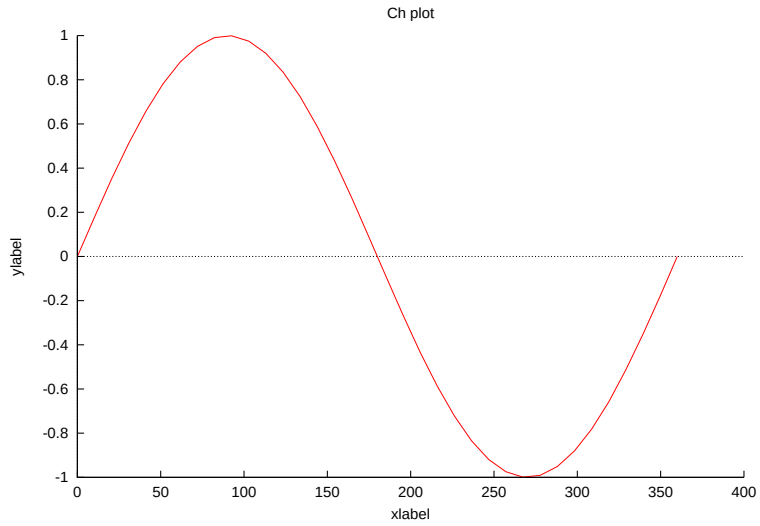
Output



Example 2

```
/* File: plotxy.ch */
/* File: plotxy.ch
   Plot data in computational arrays using function plotxu() */
#include <math.h>
#include <chplot.h>

int main() {
    int n = 37;              /* number of points for the plot */
    array double x[n], y[n]; /* arrays with data for plotting */

    lindata(0, 360, x, n);   /* fill array x with n points from 0 to 360 */
    y = sin(x*M_PI/180);     /* y = sin(x) with x in radian */
    plotxy(x, y, n, "function sin(x)", "x (degree)", "sin(x)");
    return 0;
}
```

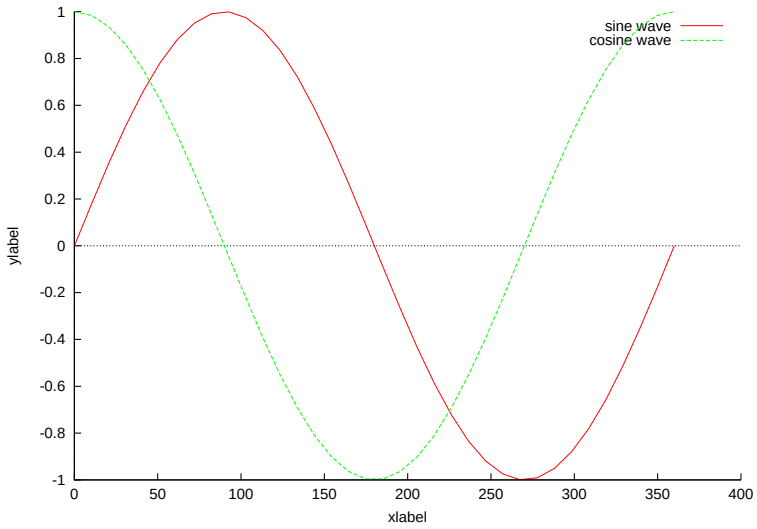
Output



Example 3

```
/* File: plotxy_2.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 36;
    array double x[numpoints], y[2][numpoints];
    int i;
    class CPlot plot;

    lindata(0, 360, x);
    for(i=0; i<numpoints; i++) {
        y[0][i] = sin(x[i]*M_PI/180);
        y[1][i] = cos(x[i]*M_PI/180);
    }
    plotxy(x, y, NULL, "xlabel", "ylabel", &plot);
    plot.legend("sine wave", 0);
    plot.legend("cosine wave", 1);
    plot.plotting();
}
```

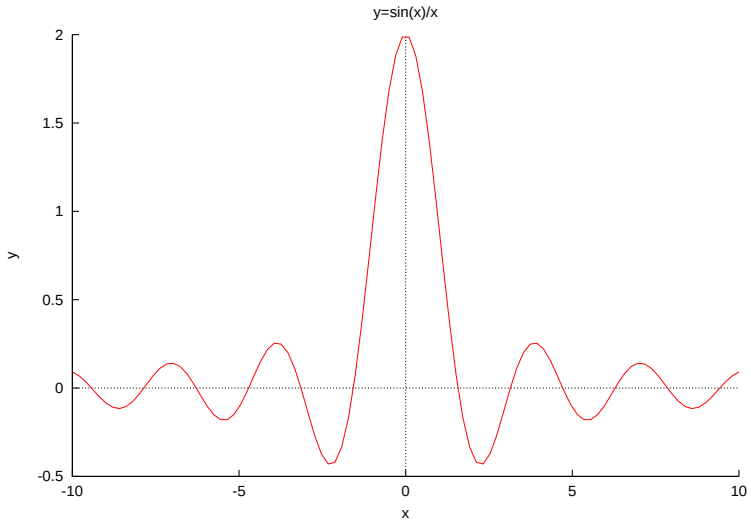
Output



Example 4

```
/* File: plotxy_4.ch */
#include <math.h>
#include <float.h>
#include <chplot.h>

int main() {
    int numpoints = 100;
    array double x[numpoints], y[numpoints];

    lindata(-10, 10, x);
    x = x+(x==0)*FLT_EPSILON; /* if x==0, x becomes epsilon */
    y = sin(2*x)./x;
    plotxy(x, y, "y=sin(x)/x", "x", "y");
}
```

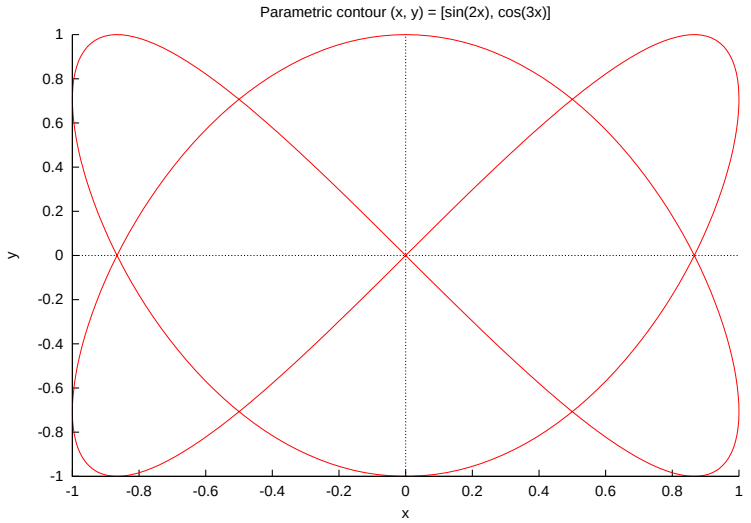
Output



Example 5
Compare with the output in the example in **CPlot**::**sizeRatio**().

```
/* File: plotxy_3.ch */
#include <math.h>
#include <chplot.h>

int main() {
    int numpoints = 360;
    array double t[numpoints], x[numpoints], y[numpoints];

    lindata(0, 2*M_PI, t);
    x = sin(2*t);
    y = cos(3*t);
    plotxy(x, y, "Parametric contour (x, y) = [sin(2x), cos(3x)]", "x", "y");
}
```

Output



See Also
CPlot, **CPlot**::**data2D**(), **CPlot**::**data2DCurve**(), **fplotxy**(), **fplotxyz**(), **plotxyf**(), **plotxyz**(), **plotxyzf**().

plotxyf

Synopsis
#include <**chplot.h**>
int plotxyf(**string_t** *file*, ... /* [**string_t** *title*, **string_t** *xlabel*, **string_t** *ylabel*], [**class CPlot** **pl*] */);

Syntax
plotxyf(*file*)
plotxyf(*file, title, xlabel, ylabel*)
plotxyf(*file, title, xlabel, ylabel, &plot*)

Purpose
Plot 2D data from a file or initialize an instance of the **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
file The file containing the data to be plotted.

title The title of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

pl A pointer to an instance of the **CPlot** class.

Description
Plot 2D data from a file or initialize a **CPlot** variable. The data file should be formatted with each data point on a separate line. 2D data are specified by two values per point. The *title*, *xlabel*, and *ylabel*, for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this function. If a non-NULL pointer is passed, it will be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting** member function. If a previously initialized **CPlot** variable is passed, it will be re-initialized with the function parameters. If no pointer or a `NULL` pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function. An empty line in the data file causes a break in the plot. Multiple curves can be plotted in this manner, however, the plot style will be the same for all curves.

The following code segment

```
class CPlot plot;
plotxyf("datafile", "title", "xlabel", "ylabel", &plot);
```

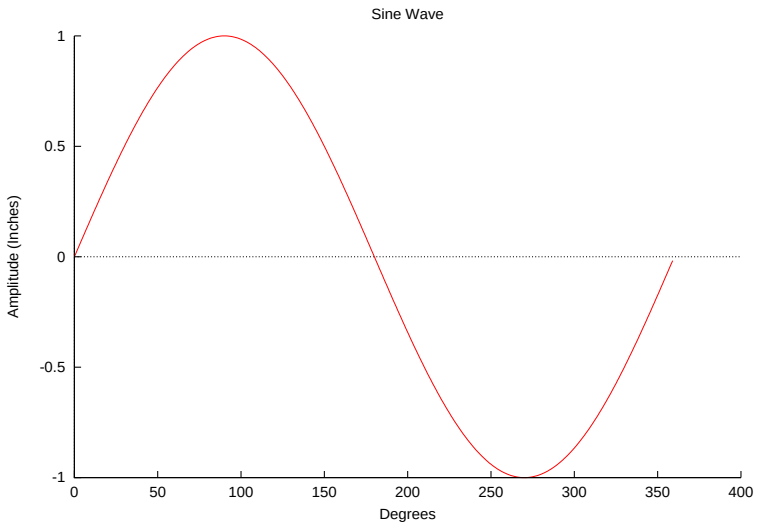
is equivalent to

```
class CPlot plot;
plot.dataFile("datafile");
plot.title("title");
plot.label(PLOT_AXIS_X, "xlabel");
plot.label(PLOT_AXIS_Y, "ylabel");
```

Example

```
/* File: plotxyf.ch */
// Demonstrates the usage of the plotxyf() function.
#include <stdio.h>
#include <chplot.h>
#include <math.h>

int main() {
    string_t title="Sine Wave",          //Define
           xlabel="Degrees",
           ylabel="Amplitude (Inches)";
    string_t file;
    file = tmpnam(NULL);                 //Create temporary file.
    int i;
    FILE *out;
    out=fopen (file,"w");              //Write data to file.
    for(i=0;i<=359;i++) fprintf(out,"%i %f \n",i,sin(i*M_PI/180));
    fclose(out);
    plotxyf(file,title,xlabel,ylabel); //Call plotting function.
    remove(file);
}
```

Output



See Also

CPlot, **CPlot**::**dataFile**, **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyzf**().

plotxyz

Synopsis
#include <**chplot.h**>
int plotxyz(**double** *x*[&], **double** *y*[&], **array double** &*z*, ... /* [int n] [int nx, int ny]
[**string_t** *title*, **string_t** *xlabel*, **string_t** *ylabel*, **string_t** *zlabel*], [**class CPlot** **pl*] */);

Syntax
plotxyz(*x*, *y*, *z*)
plotxyz(*x*, *y*, *z*, *title*, *xlabel*, *ylabel*, *zlabel*)
plotxyz(*x*, *y*, *z*, *title*, *xlabel*, *ylabel*, *zlabel*, &*plot*)
plotxyz(*x*, *y*, *z*, n)
plotxyz(*x*, *y*, *z*, n, *title*, *xlabel*, *ylabel*, *zlabel*)
plotxyz(*x*, *y*, *z*, n, *title*, *xlabel*, *ylabel*, *zlabel*, &*plot*)
plotxyz(*x*, *y*, *z*, nx, ny)
plotxyz(*x*, *y*, *z*, nx, ny, *title*, *xlabel*, *ylabel*, *zlabel*)
plotxyz(*x*, *y*, *z*, nx, ny, *title*, *xlabel*, *ylabel*, *zlabel*, &*plot*)

Purpose
Plot a 3D data set or initialize an instance of the **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x A one-dimensional array of size n_x. The value of each element is used for the x-axis of the plot.

y A one-dimensional array of size n_y. The value of each element is used for the y-axis of the plot.

z If the data are for a 3D curve, z is a m X n_z dimensional array, and $n_x = n_y = n_z$. If the data are for a 3D surface or grid, z is a m x n_z dimensional array, and $n_z = n_x \cdot n_y$.

n The number of data points for a 3D curve.

nx The number of data points in the x-coordinates for a 3D surface.

ny The number of data points in the y-coordinates for a 3D surface.

title The title of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

zlabel The y-axis label.

pl A pointer to an instance of the **CPlot** class.

Description

Plot a 3D data set or initialize a **CPlot** variable. For Cartesian data, *x* is a one-dimensional array of size n_x and *y* is a one-dimensional array of size n_y. *z* can be of two different dimensions depending on what type of data is to be plotted. If the data is for a 3D curve, *z* is a m x n_z dimensional array, and $n_x = n_y = n_z$ for the the optional argument `n`. If the data is for a 3D surface or grid, *z* is a m x n_z dimensional array, and $n_z = n_x \cdot n_y$ with optional arguments `nx` and `ny`. For cylindrical or spherical data *x* is a one dimensional array of size n_x (representing θ), *y* is a one dimensional array of size n_y (representing z or ϕ), and *z* is a m x n_z dimensional array (representing r). In all cases these data arrays can be of any supported data type. Conversion of the data to **double** type is performed internally. The *title*, *xlabel*, *ylabel*, and *zlabel* for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this function. If a non-NULL pointer is passed, it will be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting** member function. If a previously initialized **CPlot** variable is passed, it will be re-initialized with the function parameters. If no pointer or a NULL pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function.

The following code segment

```
class CPlot plot;
plotxyz(x, y, z, "title", "xlabel", "ylabel", "zlabel", &plot);
```

is equivalent to

```
class CPlot plot;
plot.data3D(x, y, z);
plot.title("title");
plot.label(PLOT_AXIS_X, "xlabel");
plot.label(PLOT_AXIS_Y, "ylabel");
plot.label(PLOT_AXIS_Z, "zlabel");
```

The following code segment for plotting a 3D curve

```
    class CPlot plot;
    plotxyz(x, y, z, n);
```

is equivalent to

```
    class CPlot plot;
    plot.data3DCurve(x, y, z, n);
```

The following code segment for plotting a 3D surface

```
    class CPlot plot;
    plotxyz(x, y, z, nx, ny);
```

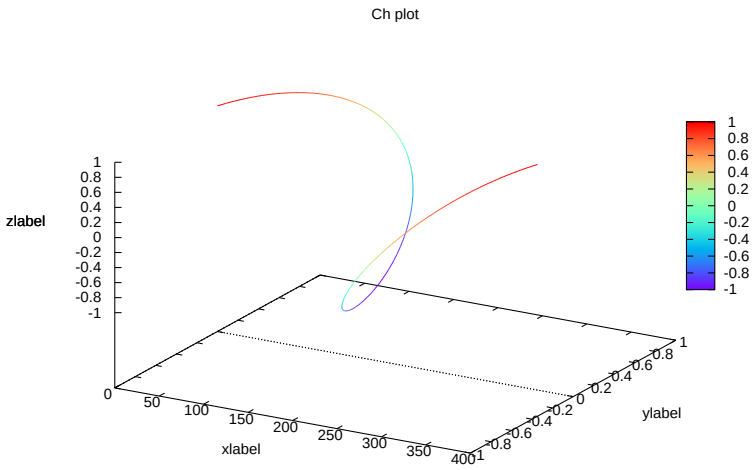
is equivalent to

```
class CPlot plot;
plot.data3DSurface(x, y, z, nx, ny);
```

Example 1

```
/* File: plotxyz.ch */
#include <math.h>
#include <chplot.h>

int main() {
    array double x[360], y[360], z[360];

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    z = cos(x*M_PI/180);
    plotxyz(x, y, z, "Ch plot", "xlabel", "ylabel", "zlabel");
    /* or
    plotxyz(x, y, z, 360, "Ch plot", "xlabel", "ylabel", "zlabel"); */
}
```

Output

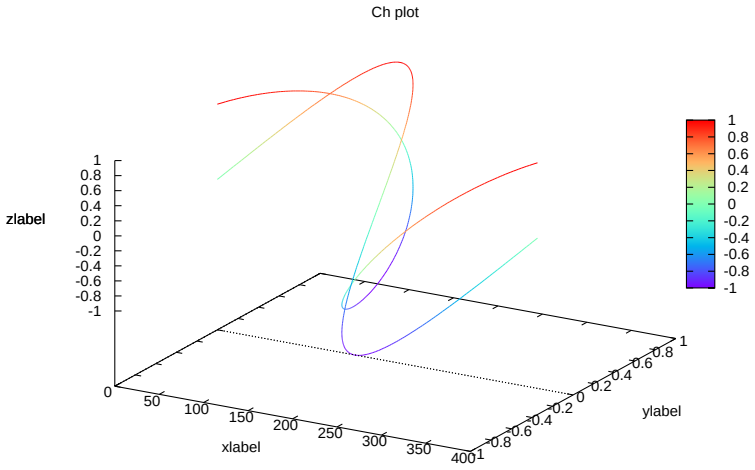


Example 2

```
/* File: plotxyz_2.ch */
#include <math.h>
#include <chplot.h>

int main() {
    array double x[360], y[360], z[2][360];
    int i;

    lindata(0, 360, x);
    y = sin(x*M_PI/180);
    for(i=0; i<360; i++) {
        z[0][i] = cos(x[i]*M_PI/180);
        z[1][i] = y[i];
```

```
    }
    plotxyz(x, y, z, "Ch plot", "xlabel", "ylabel", "zlabel");
    /* or
    plotxyz(x, y, z, 360, "Ch plot", "xlabel", "ylabel", "zlabel"); */
}
```

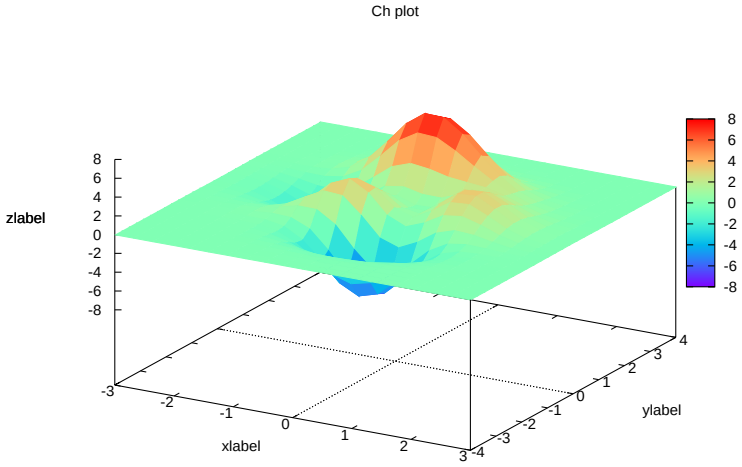
Output



Example 3

```
/* File: plotxyz_3.ch */
#include <math.h>
#include <chplot.h>

int main() {
    double x[20], y[30], z[600];
    int i,j;

    lindata(-3, 3, x);
    lindata(-4, 4, y);
    for(i=0; i<20; i++) {
        for(j=0; j<30; j++) {
            z[30*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plotxyz(x, y, z, "Ch plot", "xlabel", "ylabel", "zlabel");
    /*  or
    plotxyz(x, y, z, 20, 30, "Ch plot", "xlabel", "ylabel", "zlabel"); */
}
```

Output



See Also

CPlot, **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyf**(), **plotxyzf**().

plotxyzf

Synopsis
#include <**chplot.h**>
int plotxyzf(string_t *file*, ... /* [**string_t** *title*, **string_t** *xlabel*, **string_t** *ylabel*, **string_t** *zlabel*],
[**class CPlot** **pl*] */);

Syntax
plotxyzf(*file*)
plotxyzf(*file*, *title*, *xlabel*, *ylabel*, *zlabel*)
plotxyzf(*file*, *title*, *xlabel*, *ylabel*, *zlabel*, &*plot*)

Purpose
Plot 3D data from a file or initialize an instance of the **CPlot**class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
file The file containing the data to be plotted.

title The title of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

zlabel The z-axis label.

pl A pointer to an instance of the **CPlot** class.

Description
Plot 3D data from a file or initialize a **CPlot** variable. The data file should be formatted with each data point on a separate line. 3D data is specified by three values per data point. For a 3D grid or surface data, each row is separated in the data file by a blank line. For example, a 3 x 3 grid would be represented as follows:

```
x1   y1   z1
x1   y2   z2
x1   y3   z3

x2   y1   z4
x2   y2   z5
x2   y3   z6

x3   y1   z7
x3   y2   z8
x3   y3   z9
```

This function can only be used to plot Cartesian grid data. The *title*, *xlabel*, *ylabel*, and *zlabel* for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this function. If a non-NULL pointer is passed, it will be be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting** member function. If a previously initialized **CPlot** variable is passed, it will be re-initialized with the function parameters. If no pointer or a NULL pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function. Two empty lines in the data file will cause a break in the plot. Multiple curves or surfaces can be plotted in this manner however, the plot style will be the same for all curves or surfaces.

The following code segment

```
class CPlot plot;
plotxyzf("datafile", "title", "xlabel", "ylabel", "zlabel", &plot);
```

is equivalent to

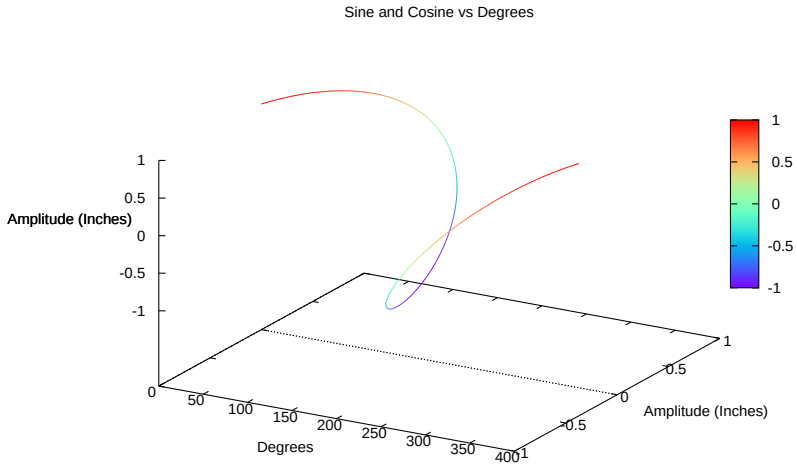
```
class CPlot plot;
plot.dimension(3);
plot.dataFile("datafile");
plot.title("title");
plot.label(PLOT_AXIS_X, "xlabel");
plot.label(PLOT_AXIS_Y, "ylabel");
plot.label(PLOT_AXIS_Z, "zlabel");
```

Example

```
/* File: plotxyzf.ch */
#include <chplot.h>
#include <stdio.h>
#include <math.h>

int main() {
    string_t title="Sine and Cosine vs Degrees",
           xlabel="Degrees",
```

```
            ylabel="Amplitude (Inches)",
            zlabel="Amplitude (Inches)";
    string_t file;
    file = tmpnam(file);                                  //temporary file
    int i;
    FILE *out;

    out=fopen (file,"w");                          // write data to file
    for(i=0;i<=359;i++) fprintf(out,"%i %f %f\n",i,sin(i*M_PI/180),cos(i*M_PI/180));
    fclose(out);
    plotxyzf(file,title,xlabel,ylabel,zlabel);  // call plotting function:
    remove(file);
}
```

Output



See Also

CPlot, **CPlot**::**dataFile**(), **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyz**().

第3章　シェル関数 — <chshell.h>

ヘッダーファイル **chshell.h** には Ch シェル関数の定義が入っています。次の関数が Ch シェルプログラミング用に設計されています。

関数

関数	説明
chinfo()	Ch 言語環境に関する情報を取得します。
iskey()	文字列がキーワードであるかどうかを調べます。
isstudent()	Ch 言語環境が Student Edition であるかどうかを調べます。
isvar()	文字列のプロパティを検出します。
sizeofelement()	配列の要素のサイズを検出します。

定数

chshell.h には次のマクロが定義されています。

定数	説明
CH_NOTVAR	0 として定義され、文字列が変数でないことを示します。
CH_SYSTEMCONST	文字列がシステム定数であることを示します。
CH_SYSTEMVAR	文字列がシステム変数であることを示します。
CH_SYSTEMFUN	文字列がシステム関数であることを示します。
CH_EXTERNVAR	文字列が外部変数であることを示します。

構造体

構造体 **chinfo_t** には次のメンバが含まれています。

Struct chinfo_t {

```
  string_t edition;          // ”Professional” or ”Student”
  string_t releasedate;      // ”October 1 1999”
  string_t version;          // ”1.0”
  unsigned int vermajor;     // 1
  unsigned int verminor;     // 0
  unsigned int vermicro;     // 0
  unsigned int verbuild;     // 0
```

}

データタイプ

データ型 **chinfo_t** は、上記の Ch 言語環境パラメータを保持します。

chinfo

構文

#include <**chshell.h**>
int chinfo(**chinfo_t** **info*);

目的

Ch 言語環境に関する情報を取得します。

戻り値

この関数は、成功した場合は 0 を、失敗した場合は-1 を返します。

パラメータ

info info エディション、リリース年月日、バージョン、メジャーバージョン番号、マイナーバージョン番号、マイクロバージョン番号、ビルド番号を保持する構造体へのポインタ。

説明

この関数は、エディション、リリース年月日、バージョン、メジャーバージョン番号、マイナーバージョン番号、マイクロバージョン番号、ビルド番号などの Ch 言語環境に関する情報を取得します。

例

```
#include <chshell.h>

int main() {
  chinfo_t info;

  if(chinfo(&info)==-1)
     printf("Error: chinfo() error\n");
  printf("info.edition  = %s\n", info.edition);
  printf("info.releasedate = %s\n", info.releasedate);
  printf("info.version  = %s\n", info.version);
  printf("info.vermajor = %d\n", info.vermajor);
  printf("info.verminor = %d\n", info.verminor);
  printf("info.vermicro = %d\n", info.vermicro);
  printf("info.verbuild = %d\n", info.verbuild);
  printf("Ch %s Edition v%d.%d.%d.%d %s released by %s\n", info.edition,
         info.vermajor, info.verminor, info.vermicro, info.verbuild,
         info.version, info.releasedate);
}
```

出力

```
info.edition  = Professional
info.releasedate = March 1, 2007
info.version  =
info.vermajor = 5
info.verminor = 5
info.vermicro = 0
info.verbuild = 13151
Ch Professional Edition v5.5.0.13151  released by March 1, 2007
```

関連項目
isstudent().

iskey

構文

#include <**chshell.h**>
int iskey(string_t *name*);

目的

文字列がキーワードであるかどうかを調べます。

戻り値

この関数は、文字列が汎用関数である場合は 1 を、英字で始まるキーワードである場合は 2 を、それ以外のキーワードである場合は 3 を、その他の場合は 0 を返します。

パラメータ

name 入力文字列。

説明

この関数は、入力文字列 *name* がキーワードであるかどうかを判断します。

例

```
/* access element of a string */
#include <stdio.h>
#include <string.h>
#include <chshell.h>

int g;
int main() {
  int i;

  printf("iskey(\"sin\") = %d\n", iskey("sin"));
  printf("iskey(\"int\") = %d\n", iskey("int"));
  printf("iskey(\"=\") = %d\n", iskey("="));
  printf("iskey(\"unknown\") = %d\n", iskey("unknown"));

  printf("isstudent() = %d\n", isstudent());

  printf("isvar(\"i\") == CH_NOTVAR = %d\n", isvar("i") == CH_NOTVAR);
  printf("isvar(\"NaN\") == CH_SYSTEMCONST = %d\n", isvar("NaN") == CH_SYSTEMCONST);
  printf("isvar(\"_path\") == CH_SYSTEMVAR = %d\n", isvar("_path") == CH_SYSTEMVAR);
  printf("isvar(\"g\") == CH_EXTERNVAR = %d\n", isvar("g") == CH_EXTERNVAR);
  printf("isvar(\"isvar\") == CH_EXTERNFUN = %d\n", isvar("isvar") == CH_EXTERNFUN);
  printf("isvar(\"unknown\") = %d\n", isvar("unknown"));
}
```

出力

```
iskey("sin") = 1
iskey("int") = 2
iskey("=") = 3
iskey("unknown") = 0
```

```
isstudent() = 0
isvar("i") == CH_NOTVAR = 1
isvar("NaN") == CH_SYSTEMCONST = 1
isvar("_path") == CH_SYSTEMVAR = 1
isvar("g") == CH_EXTERNVAR = 1
isvar("isvar") == CH_EXTERNFUN = 1
isvar("unknown") = 0
```

関連項目

isenv(), **isnum**(), **isstudent**(), **isvar**().

isstudent

構文
#include <**chshell.h**>
int isstudent(**void**);

目的
Ch 言語環境が Student Edition であるかどうかを調べます。

戻り値
この関数は、Ch 言語環境が Student Edition である場合は 1 を、Professional Edition である場合は 0 を返します。

説明
この関数は、現在の Ch 言語環境が Student Edition であるかどうかを判断します。

例
iskey() を参照。

関連項目
isenv(), **isnum**(), **iskey**(), **isvar**().

---

isvar

構文
#include <**chshell.h**>
int isvar(**string_t** *name*);

目的
文字列のプロパティを検出します。

戻り値
この関数は、次のいずれか 1 つの値を返します。

CH_NOTVAR 文字列は変数でない。

CH_SYSTEMCONST 文字列はシステム定数である。

CH_SYSTEMVAR 文字列はシステム変数である。

CH_SYSTEMFUN 文字列はシステム関数である。

CH_EXTERNVAR 文字列は外部変数である。

パラメータ
name 入力文字列。

説明
この関数は、入力文字列 *name* の特性を示す整数を返します。

例
iskey() を参照。

関連項目
isenv(), **iskey**(), **isnum**(), **isstudent**().

sizeofelement

構文

#include <chshell.h>
int sizeofelement(int *etype*);

目的

配列の要素のサイズを検出します。

戻り値

この関数は、整数値表現でデータ型の文字数を返します。

Parameters

etype 配列の要素のデータ型を示す整数。

説明

この関数は、配列の要素のサイズを検出します。引数 *etype* は配列のデータ型であり、組み込み関数 **elementtype()** を呼び出すことで取得できます。

例

```
/* sizeofelement() returns the size of the data type of the
   its argument. If it is an array, the data type of the array element
   is used */

#include <stdio.h>

void func1(double a[&]) {
    int size;

    size =sizeofelement(elementtype(a));
    printf("sizeofelement(a) = %d\n", size);
}

void func2(array double &a) {
    int size;

    size =sizeofelement(elementtype(a));
    printf("sizeofelement(a) = %d\n", size);
}
int main() {
    int    a1[3][4], b1[3];
    double a2[3][4], b2[3];
    func1(b1);
    func1(b2);
    func2(a1);
    func2(b1);
    func2(a2);
    func2(b2);
}
```

出力

```
sizeofelement(a) = 4
sizeofelement(a) = 8
sizeofelement(a) = 4
sizeofelement(a) = 4
sizeofelement(a) = 8
sizeofelement(a) = 8
```

関連項目

elementtype().

第4章　複素関数 — <complex.h>

ヘッダー **complex.h** では、最新の ISO C 標準における複素計算をサポートするマクロを定義し、関数を宣言しています。各構文は、少なくとも 1 つの double complex 型パラメータ、double complex 型または double 型の数、戻り値をもった主関数、および同じ関数名でその最後に f および l のサフィックスが付き、それぞれ float 型と long double 型パラメータと戻り値をもつ関数に対応する別関数から構成されます。

関数

以下の関数が **complex.h** ヘッダーファイル内で宣言されています。

関数	説明
cabs()	複素数の絶対値を計算します。
cacos()	複素数のアークコサインを計算します。
cacosh()	複素数の逆双曲線コサインを計算します。
carg()	複素数の引数を計算します。
casin()	複素数のアークサインを計算します。
casinh()	複素数の逆双曲線サインを計算します。
catan()	複素数のアークタンジェントを計算します。
catanh()	複素数の逆双曲線タンジェントを計算します。
ccos()	複素数のコサインを計算します。
ccosh()	複素数の双曲線コサインを計算します。
cexp()	複素数の指数関数を計算します。
cimag()	複素数の虚部を取得します。
clog()	複素数の対数を計算します。
conj()	複素共役を計算します。
cpow()	2 つの変数の複素数べき乗値を計算します。
creal()	複素数の実部を取得します。
csin()	複素数のサインを計算します。
csinh()	複素数の双曲線サインを計算します。
ctan()	複素数のタンジェントを計算します。
ctanh()	複素数の双曲線タンジェントを計算します。
iscnan()	引数が複素非数であるかどうかをテストします。

マクロ

以下のマクロが **complex.h** ヘッダーファイルで定義されています。

マクロ	説明
I	虚数 I が **complex(0.0, 1.0)** に展開されます。

移植性

このヘッダーには移植性に関する既知の問題はありません。

C と Ch の差異

関数 **conj** は Ch の内蔵関数です。

cabs

Synopsis
#include <**complex.h**>
double cabs(**double complex** *z*);
float cabsf(**float complex** *z*);
long double cabsl(**long double complex** *z*);

Purpose
Calculates an absolute value of a complex number.

Return Value
The **cabs** functions return the complex absoulte value.

Parameters
z Complex argument.

Description
The **cabs** functions compute the complex absolute value (also called norm, modulus or magnitude) of z.

It is recommended that the polymorphic generic function **abs**(), instead of the **cabs**() function, be used to compute the complex absolute value of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("cabs(z) = %f\n", cabs(z));
}
```

Output

```
cabs(z) = 2.236068
```

See Also
abs().

cacos

Synopsis
#include <complex.h>
double complex cacos(double complex *z*);
float complex cacosf(float complex *z*);
long double complex cacosl(long double complex *z*);

Purpose
Calculates an arc cosine of a complex number.

Return Value
The **cacos** functions return the complex arc cosine value, in the range of a strip mathematically unbound along the imaginary axis, and in the interval [0, π] along the real axis.

Parameters
z Complex argument.

Description
The **cacos** functions compute the complex arc cosine of *z*, with branch cuts outside the interval [-1, 1] along the real axis.

It is recommended that the polymorphic generic function **acos()** instead of the **cacos**() function, be used to compute the complex arc cosine of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("cacos(z) = %f\n", cacos(z));
}
```

Output

```
cacos(z) = complex(1.143718,-1.528571)
```

See Also
cos(), **ccos**(), **cacosh**(), **ccosh**().

cacosh

Synopsis
#include <complex.h>
double complex cacosh(double complex *z*);
float complex cacoshf(float complex *z*);
long double complex cacoshl(long double complex *z*);

Purpose
Calculates an arc hyperbolic cosine of a complex number.

Return Value
The **cacosh** functions return the complex arc hyperbolic cosine value, in the range of a half-strip of non-negative values along the real axis, and in the interval [-$i\pi$, $i\pi$] along the imaginary axis.

Parameters
z Complex argument.

Description
The **cacosh** functions compute the complex arc cosine of *z*, with branch a cut at values less than 1 along the real axis.

It is recommended that the polymorphic generic function **acosh()**, instead of the **cacosh**() function, be used to compute the complex arc hyperbolic cosine of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("cacosh(z) = %f\n", cacosh(z));
}
```

Output

```
cacosh(z) = complex(1.528571,1.143718)
```

See Also
cos(), **cacos**(), **ccos**(), **ccosh**().

carg

Synopsis
#include <**complex.h**>
double carg(**double complex** *z*);
float cargf(**float complex** *z*);
long double cargl(**long double complex** *z*);

Purpose
Calculates an argument of a complex number.

Return Value
The **carg** functions return the value of the argument in the range [-π, π].

Parameters
z Complex argument.

Description
The **carg** functions compute the argument (also called phase angle) of z, with a branch cut along the negative real axis.

It is recommended that the polymorphic generic function **arg()**, instead of the **carg**() function, be used to compute the argumentof a complex number *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
  int i=7;
  double f = 30;
  double d = -10;
  complex z = complex(3, 4);
  printf("carg(%d) = %f radian \n", i, carg(i));
  printf("carg(%f) = %f radian \n", f, carg(f));
  printf("carg(%lf) = %f radian \n", d, carg(d));
  printf("carg(%f) = %f radian \n", z, carg(z));
}
```

Output

```
carg(7) = 0.000000 radian
carg(30.000000) = 0.000000 radian
carg(-10.000000) = 3.141593 radian
carg(complex(3.000000,4.000000)) = 0.927295 radian
```

See Also

casin

Synopsis
#include <**complex.h**>
double complex casin(**double complex** *z*);
float complex casinf(**float complex** *z*);
long double complex casinl(**long double complex** *z*);

Purpose
Calculates an arc sine of a complex number.

Return Value
The **casin** functions return the complex arc sine value, in the range of a strip mathematically unbounded along the imaginary axis, and in the interal [-π/2, π/2] along the real axis.

Parameters
z Complex argument.

Description
The **casin** functions compute the complex arc sine of *z*, with branch cuts outside the interval [-1, 1] along the real axis.

It is recommended that the polymorphic generic function **asin()**, instead of the **casin**() function, be used to compute the complex arc sine of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("casin(z) = %f\n", casin(z));
}
```

Output

```
casin(z) = complex(0.427079,1.528571)
```

See Also
sin(), **csin**(), **casinh**(), **csinh**().

casinh

Synopsis
#include <**complex.h**>
double complex casinh(**double complex** *z*);
float complex casinhf(**float complex** *z*);
long double complex casinhl(**long double complex** *z*);

Purpose
Calculates an arc hyperbolic sine of a complex number.

Return Value
The **casinh** functions return the complex arc hyperbolic sine value, in the range of a strip mathematically unbounded along the real axis, and in the interval [-$i\pi$/2, $i\pi$/2] along the imaginary axis.

Parameters
z Complex argument.

Description
The **casinh** functions compute the complex arc hyperbolic sine of *z*, with branch cuts outside the interval $[-i, i]$ along the imaginary axis.

It is recommended that the polymorphic generic function **asinh()**, instead of the **casinh**() function, be used to compute the complex arc hyperbolic sine of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("casinh(z) = %f\n", casinh(z));
}
```

Output

```
casinh(z) = complex(1.469352,1.063440)
```

See Also
sin(), **casin**(), **csin**(), **csinh**().

catan

Synopsis
#include <**complex.h**>
double complex catan(**double complex** *z*);
float complex catanf(**float complex** *z*);
long double complex catanl(**long double complex** *z*);

Purpose
Calculates an arc tangent of a complex number.

Return Value
The **catan** functions return the complex are tangent value, in the range of a strip mathematically unbounded along the imaginary axis, and in the interval [-π/2, π/2].

Parameters
z Complex argument.

Description
The **catan** functions compute the complex arc tangent of *z*, with branch cuts outside the interval [-*i*, *i*] along the imaginary axis.

It is recommended that the polymorphic generic **atan()** function, instead of the **catan**() function, be used to compute the complex arc tangent of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("catan(z) = %f\n", catan(z));
}
```

Output

```
catan(z) = complex(1.338973,0.402359)
```

See Also
tan(), **ctan**(), **catanh**(), **ctanh**().

catanh

Synopsis
#include <**complex.h**>
double complex catanh(**double complex** *z*);
float complex catanhf(**float complex** *z*);
long double complex catanhl(**long double complex** *z*);

Purpose
Calculates an arc hyperbolic tangent of a complex number.

Return Value
The **catanh** functions reutrn the complex arc hyperbolic tangent value, in the range of a strip mathematically unbounded along the real axis, and in the interval [-*i*π/2, *i*π/2] along the imaginary axis.

Parameters
z Complex argument.

Description
The **catanh** functions compute the complex arc hyperbolic tangent of *z*, with branch cuts outside the interval [-1, 1] along the real axis.

It is recommended that the polymorphic generic **atanh()** function, instead of the **catanh**() functions, be used to compute the complex arc hyperbolic tangent of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("catanh(z) = %f\n", catanh(z));
}
```

Output

```
catanh(z) = complex(0.173287,1.178097)
```

See Also
tan(), **catan**(), **ctan**(), **ctanh**().

CCOS

Synopsis

#include <**complex.h**>
double complex ccos(**double complex** *z*);
float complex ccosf(**float complex** *z*);
long double complex ccosl(**long double complex** *z*);

Purpose

Calculates a cosine of a complex number.

Return Value

The **ccos** functions return the complex cosine.

Parameters

z Complex argument.

Description

The **cacos** functions compute the complex cosine of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("ccos(z) = %f\n", ccos(z));
}
```

Output

```
ccos(z) = complex(2.032723,-3.051898)
```

See Also

cos(), **cacos**(), **cacosh**(), **ccosh**().

ccosh

Synopsis
#include <**complex.h**>
double complex ccosh(**double complex** *z*);
float complex ccoshf(**float complex** *z*);
long double complex ccoshl(**long double complex** *z*);

Purpose
Calculates a hyperbolic cosine of a complex number.

Return Value
The **ccosh** functions return the complex hyperbolic cosine.

Parameters
z Complex argument.

Description
The **cacosh** functions compute the complex hyperbolic cosine of z.

It is recommended that the polymorphic generic function **cosh()**, instead of the **ccosh**() function, be used to compute the complex hyperbolic cosine of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("ccosh(z) = %f\n", ccosh(z));
}
```

Output

```
ccosh(z) = complex(-0.642148,1.068607)
```

See Also
cos(), **cacos**(), **ccos**(), **cacosh**().

cexp

Synopsis
#include <**complex.h**>
double complex cexp(**double complex** *z*);
float complex cexpf(**float complex** *z*);
long double complex cexpl(**long double complex** *z*);

Purpose
Calculates a base-*e* exponential of a complex number.

Return Value
The **cexp** functions return the complex base-*e* exponential value.

Parameters
z Complex argument.

Description
The **cexp** functions compute the complex base-*e* exponential of z.

It is recommended that the polymorphic generic function **exp()**, instead of the **cexp**() function, be used to compute the base-*e* exponential of complex number *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("cexp(z) = %f\n", cexp(z));
}
```

Output

```
cexp(z) = complex(-1.131204,2.471727)
```

See Also
exp().

cimag

Synopsis
#include <**complex.h**>
double complex cimag(double complex *z*);
float complex cimagf(float complex *z*);
long double complex cimagl(long double complex *z*);

Purpose
Get the imaginary part of a complex number.

Return Value
The **cimag** functions return the imaginary part value (as a real).

Parameters
z Complex argument.

Description
The **cimag** functions compute the imaginary part of z.

It is recommended that the polymorphic generic function **imag**(), instead of the **cimag**() function, be used to obtain the imaginary part of a complex number *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("cimag(z) = %f\n", cimag(z));
}
```

Output

```
cimag(z) = 2.000000
```

See Also
imag().

clog

Synopsis
#include <**complex.h**>
double complex clog(**double complex** *z*);
float complex clogf(**float complex** *z*);
long double complex clogl(**long double complex** *z*);

Purpose
Calculates a natural base-*e* logarithm of a complex number.

Return Value
The **clog** functions return the complex natural base-*e* logarithm value, in the range of a strip mathematically unbounded along the real axis, and in the interal [-iπ, iπ] along the imaginary axis.

Parameters
z Complex argument.

Description
The **clog** functions compute the complex natural base-*e* logarithm of z, with a branch cut along the negative real axis.

It is recommended that the polymorphic generic **log()** functions, instead of the **clog**() functions, be used to compute the natural base-*e* logarithm of the complex number z.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("clog(z) = %f\n", clog(z));
}
```

Output

```
clog(z) = complex(0.628609,1.107149)
```

See Also
log().

conj

Synopsis
#include <**complex.h**>
double complex conj(**double complex** *z*);
float complex conjf(**float complex** *z*);
long double complex conjl(**long double complex** *z*);

Purpose
Calculates the conjugate of a complex number.

Return Value
The **conj** functions return the complex conjugate value.

Parameters
z Complex argument.

Description
The **conj** functions compute the complex conjugate of z, by reversing the sign of its imaginary part.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("conj(z) = %f\n", conj(z));
}
```

Output

```
conj(z) = complex(1.000000,-2.000000)
```

See Also

cpow

Synopsis
#include <**complex.h**>
double complex cpow(**double complex** *z*, **double complex** *y*);
float complex cpowf(**float complex** *z*, **float complex** *y*);
long double complex cpowl(**long double complex** *z*,**long double** complex *y*);

Purpose
Calculates the power function of a complex number.

Return Value
The **cpow** functions return the complex power function value.

Parameters
z Complex argument.

y Complex argument.

Description
The **cpow** functions compute the complex power function x^y, with a branch cut for the first parameter along the negative real axis.

It is recommended that the polymorphic generic function **pow()**, instead of the **cpow**() function, be used to compute the power function of a complex number *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   double complex y = complex(3, 4);
   printf("cpow(z,y) = %f\n", cpow(z,y));
}
```

Output

```
cpow(z,y) = complex(0.129010,0.033924)
```

See Also
pow().

creal

Synopsis
#include <**complex.h**>
double creal(**double complex** *z*);
float crealf(**float complex** *z*);
long double creall(**long double complex** *z*);

Purpose
Get the real part of a complex number.

Return Value
The **creal** functions return the real part value.

Parameters
z Complex argument.

Description
The **creal** functions compute the real part of *z*.

It is recommended that the polymorphic generic function **real**(), instead of the **creal**() function, be used to compute the real part of a complex number *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("conj(z) = %f\n", conj(z));
}
```

Output

```
conj(z) = complex(1.000000,-2.000000)
```

See Also
real().

csin

Synopsis
#include <**complex.h**>
double complex csin(**double complex** *z*);
float complex csinf(**float complex** *z*);
long double complex csinl(**long double complex** *z*);

Purpose
Calculates a sine of a complex number.

Return Value
The **csin** functions return the complex sine.

Parameters
z Complex argument.

Description
The **csin** functions compute the complex sine of *z*.

It is recommended that the polymorphic generic **sin()** functions, instead of the **csin**() functions, be used to compute the complex sine of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("csin(z) = %f\n", csin(z));
}
```

Output

```
csin(z) = complex(3.165779,1.959601)
```

See Also
sin(), **casin**(), **casinh**(), **csinh**().

csinh

Synopsis
#include <**complex.h**>
double complex csinh(**double complex** *z*);
float complex csinhf(**float complex** *z*);
long double complex csinhl(**long double complex** *z*);

Purpose
Calculates a hyperbolic sine of a complex number.

Return Value
The **csinh** functions return the complex hyperbolic sine.

Parameters
z Complex argument.

Description
The **csinh** functions compute the complex hyperbolic sine of z.

It is recommended that the polymorphic generic **sinh()** functions, instead of the **csinh**() functions, be used to compute the complex hyperbolic sine of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("csinh(z) = %f\n", csinh(z));
}
```

Output

```
csinh(z) = complex(-0.489056,1.403119)
```

See Also
sin(), **casin**(), **csin**(), **casinh**().

csqrt

Synopsis
#include <**complex.h**>
double complex csqrt(**double complex** *z*);
float complex csqrtf(**float complex** *z*);
long double complex csqrtf(**long double complex** *z*);

Purpose
Calculates a square root of a complex number.

Return Value
The **csqrt** functions return the complex square root value, in the range of the right half plane (including the imaginary axis).

Parameters
z Complex argument.

Description
The **csqrt** functions compute the complex complex square root of z, with a branch cut clong the negative real axis.

It is recommended that the polymorphic generic **sqrt()** functions, instead of the **csqrt**() function, be used to compute the complex square root of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("csqrt(z) = %f\n", csqrt(z));
}
```

Output

```
csqrt(z) = complex(1.272020,0.786151)
```

See Also

ctan

Synopsis
#include <**complex.h**>
double complex ctan(double complex *z*);
float complex ctanf(float complex *z*);
long double complex ctanl(long double complex *z*);

Purpose
Calculates a tangent of a complex number.

Return Value
The **ctan** functions return the complex tangent.

Parameters
z Complex argument.

Description
The **ctan** functions compute the complex tangent of *z*.

It is recommended that the polymorphic generic **tan()** functions, instead of the **ctan**() functions, be used to compute the complex tangent of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("ctan(z) = %f\n", ctan(z));
}
```

Output

```
ctan(z) = complex(0.033813,1.014794)
```

See Also
tan(), **catan**(), **catanh**(), **ctanh**().

ctanh

Synopsis
#include <**complex.h**>
double complex ctanh(**double complex** *z*);
float complex ctanhf(**float complex** *z*);
long double complex ctanhl(**long double complex** *z*);

Purpose
Calculates a hyperbolic tangent of a complex number.

Return Value
The **ctanh** functions return the complex hyperbolic tangent.

Parameters
z Complex argument.

Description
The **ctanh** functions compute the complex hyperbolic tangent of z.

It is recommended that the polymorphic generic **tanh()** functions, instead of the **ctanh**() functions, be used to compute the complex hyperbolic tangent of *z*.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z = complex(1, 2);
   printf("ctanh(z) = %f\n", ctanh(z));
}
```

Output

```
ctanh(z) = complex(1.166736,-0.243458)
```

See Also
tan(), **catan**(), **ctan**(), **catanh**().

iscnan

Synopsis
#include <**complex.h**>
int iscnan(complex *z*);

Purpose
Tests whether the complex number is a complex Not-a-Number, ComplexNaN.

Return Value
The **iscnan** function returns 1 value when *z* is a complex Not-a-Number.

Parameters
z complex number.

Description
The **iscnan** function tests whether the complex number is a complex Not-a-Number.

Example

```
#include <complex.h>
#include <stdio.h>

int main () {
   double complex z1 = complex(1, 2);
   double complex z2 = ComplexNaN;
   complex z3 = complex(1,2);
   complex z4 = ComplexNaN;
   printf("iscnan(z1) = %d\n", iscnan(z1));
   printf("iscnan(z2) = %d\n", iscnan(z2));
   printf("iscnan(z3) = %d\n", iscnan(z3));
   printf("iscnan(z4) = %d\n", iscnan(z4));
   printf("iscnan(ComplexNaN) = %d\n", iscnan(ComplexNaN));
   printf("iscnan(ComplexInf) = %d\n", iscnan(ComplexInf));
}
```

Output

```
iscnan(z1) = 0
iscnan(z2) = 1
iscnan(z3) = 0
iscnan(z4) = 1
iscnan(ComplexNaN) = 1
iscnan(ComplexInf) = 0
```

See Also

第5章　文字の処理 — <ctype.h>

ヘッダー **ctype.h** には、文字のテストとマッピングに役立つさまざまな関数が宣言されています。引数が int 型である場合は常に、引数の値は **unsigned character** として表現可能であるか、またはヘッダーファイル **stdio.h** に定義されたファイルの終わりを表すマクロ **EOF** の値に等しくなければなりません。引数がこれ以外の値である場合の動作は不定です。これらの関数は、true の場合はゼロ以外の値、false の場合はゼロを返します。

これらの関数の動作は、現在のロケールによる影響を受けます。“C”ロケール以外の場合のみ、ロケール固有の動作を行う関数を次に示します。出力文字という用語はロケール固有文字セットのメンバを指し、各文字はディスプレイデバイス上で 1 文字分の出力位置を占有します。制御文字という用語は出力文字ではないロケール固有文字セットのメンバを指します。

関数

関数	説明
isalnum()	引数が数字またはアルファベットであるかどうかをテストします。
isalpha()	引数がアルファベットであるかどうかをテストします。
iscntrl()	引数が制御文字であるかどうかをテストします。
isdigit()	引数が 10 進数字であるかどうかをテストします。
isgraph()	引数が出力文字であるかどうかをテストします。
islower()	引数がアルファベット小文字であるかどうかをテストします。
isprint()	引数が出力文字であるかどうかをテストします。
ispunct()	引数が句読点文字であるかどうかをテストします。
isspace()	引数が空白文字であるかどうかをテストします。
isupper()	引数がアルファベット大文字であるかどうかをテストします。
isxdigit()	引数が 16 進数字であるかどうかをテストします。
tolower()	アルファベット大文字を小文字に変換します。
toupper()	アルファベット小文字を大文字に変換します。

移植性

このヘッダーには移植性に関する既知の問題はありません。

isalnum

構文

#include <**ctype.h**>
int isalnum(int *c*);

目的

文字が数字またはアルファベットであるかどうかをテストします。

戻り値

isalnum 関数は、*c* が数字またはアルファベットならばゼロ以外の値を返します。

パラメータ

c テストされる文字。

説明

isalnum 関数は、 **isalpha** または **isdigit** が true である任意の文字をテストします

例

```
/* a sample program testing isalnum() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c='A';
    printf("isalnum('%c') returns %d\n", c,isalnum(c));
    if(isalnum(c) > 0)
        printf("'%c' is an alphanumeric character.\n",c);
    else
        printf("'%c' is not an alphanumeric character.\n", c);
    c = '4';
    printf("isalnum('%c') returns %d\n", c, isalnum(c));
    if(isalnum(c) > 0)
        printf("'%c' is an alphanumeric character.\n",c);
    else
        printf("'%c' is not an alphanumeric character.\n", c);

}
```

出力

```
isalnum('A') returns 1
'A' is an alphanumeric character.
isalnum('4') returns 1
'4' is an alphanumeric character.
```

関連項目

isalpha

構文

#include <**ctype.h**>
int isalpha(**int** *c*);

目的

文字がアルファベットであるかどうかをテストします。

戻り値

isalpha 関数は、*c* がアルファベットならばゼロ以外の値を返します。

パラメータ

c テストされる文字。

説明

isalpha 関数は、**isupper** または **islower** が true である任意の文字、または **iscntrl**、**isdigit** 、 **ispunct** 、または **isspace** のいずれもが true でないロケール固有のアルファベット文字セットの文字である任意の文字をテストします。“C”ロケールでは、**isalpha** は、 **isupper** または **islower** が true の場合のみ true を返します。

例

```
/* a sample program testing isalpha() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c='A';
    printf("isalpha('%c') returns %d\n", c, isalpha(c));
    if(isalpha(c) > 0)
        printf("'%c' is an alphabetical character.\n", c);
    else
        printf("'%c' is not an alphabetical character.\n",c);
    c =' ';
    printf("isalpha('%c') returns %d\n", c,isalpha(c));
    if(isalpha(c) > 0)
        printf("'%c' is an alphabetical character.\n", c);
    else
        printf("'%c' is not an alphabetical character.\n",c);

}
```

出力

```
isalpha('A') returns 1
'A' is an alphabetical character.
isalpha(' ') returns 0
' ' is not an alphabetical character.
```

関連項目

iscntrl

構文

#include <**ctype.h**>
int iscntrl(int *c*);

目的

文字が制御文字であるかどうかをテストします。

戻り値

iscntrl 関数は、*c* が制御文字ならばゼロ以外の値を返します。

パラメータ

c テストされる文字。

説明

iscntrl 関数は、任意の制御文字をテストします。

例

```
/* a sample program testing iscntrl() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c = 'A';
    printf("iscntrl('%c') returns %d\n", c, iscntrl(c));
    if(iscntrl(c) > 0)
        printf("'%c' is a control character.\n", c);
    else
        printf("'%c' is not a control character.\n",c);
}
```

出力

```
iscntrl('A') returns 0
'A' is not a control character.
```

関連項目

isdigit

構文

#include <**ctype.h**>
int isdigit(int *c*);

目的

文字が 10 進数字であるかどうかをテストします。

戻り値

isdigit 関数は、*c* が 10 進数字ならばゼロ以外の値を返します。

パラメータ

c テストされる文字。

説明

isdigit 関数は、任意の 10 進数字をテストします。

例

```
/* a sample program testing isdigit() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c = 'A';
    printf("isdigit('%c') returns %d\n", c, isdigit(c));
    if(isdigit(c) > 0)
        printf("'%c' is a digit.\n", c);
    else
        printf("'%c' is not a digit.\n",c);
    c = '1';
    printf("isdigit('%c') returns %d\n", c, isdigit(c));
    if(isdigit(c) > 0)
        printf("'%c' is a digit.\n",c);
    else
        printf("'%c' is not a digit.\n",c);

}
```

出力

```
isdigit('A') returns 0
'A' is not a digit.
isdigit('1') returns 1
'1' is a digit.
```

関連項目

isgraph

構文

#include <**ctype.h**>
int isgraph(int *c*);

目的

文字が出力文字であるかどうかをテストします。

戻り値

isgraph 関数は、*c* が出力文字ならばゼロ以外の値を返します。
Parameters
c テストされる文字。

説明

isgraph 関数は、スペース **(' ')** を除く任意の出力文字をテストします。

例

```
/* a sample program testing isgraph() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c = 'A';
    printf("isgraph('%c') returns %d\n", c, isgraph(c));
    if(isgraph(c) > 0)
        printf("'%c' is a printable character.\n", c);
    else
        printf("'%c' is not a printable character or it is a space.\n", c);
    c = ' ';
    printf("isgraph('%c') returns %d\n", c, isgraph(c));
    if(isgraph(c) > 0)
        printf("'%c' is a printable character.\n", c);
    else
        printf("'%c' is not a printable character or it is a space.\n", c);
}
```

出力

```
isgraph('A') returns 1
'A' is a printable character.
isgraph(' ') returns 0
' ' is not a printable character or it is a space.
```

関連項目

islower

構文

#include <**ctype.h**>
int islower(int *c*);

目的

文字がアルファベット小文字であるかどうかをテストします。

戻り値

islower 関数は、*c* がアルファベット小文字ならばゼロ以外の値を返します。

パラメータ

c テストされる文字。

説明

islower 関数は、アルファベット小文字である、または **iscntrl** 、 **isdigit** 、 **ispunct** 、または **isspace** のいずれもが true でないロケール固有のアルファベット文字セットの文字である任意の文字をテストします。“C”ロケールでは、**islower** は、英小文字であると定義された文字に対してのみ true を返します。

例

```
/* a sample program testing islower() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c = 'A';
    printf("islower('%c') returns %d\n", c,islower(c));
    if(islower(c) > 0)
        printf("'%c' is a lower case character.\n", c);
    else
        printf("'%c' is not a lower case character.\n", c);
    c = 'a';
    printf("islower('%c') returns %d\n", c,islower(c));
    if(islower(c) > 0)
        printf("'%c' is a lower case character.\n", c);
    else
        printf("'%c' is not a lower case character.\n",c);

}
```

出力

```
islower('A') returns 0
'A' is not a lower case character.
islower('a') returns 1
'a' is a lower case character.
```

関連項目

isprint

構文

#include <**ctype.h**>
int isprint(int *c*);

目的

文字が出力文字であるかどうかをテストします。

戻り値

isprint 関数は、*c* が出力文字ならばゼロ以外の値を返します。

パラメータ

c テストされる文字。

説明

isprint 関数は、スペース (**' '**) を含む任意の出力文字をテストします。

例

```
/* a sample program testing isprint() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c = 'A';
    printf("isprint('%c') returns %d\n", c,isprint(c));
    if(isprint(c) > 0)
        printf("'%c' is a printable character.\n", c);
    else
        printf("'%c' is not a printable character.\n", c);
    c = ' ';
    printf("isprint('%c') returns %d\n", c,isprint(c));
    if(isprint(c) > 0)
        printf("'%c' is a printable character.\n", c);
    else
        printf("'%c' is not a printable character.\n",c);

}
```

出力

```
isprint('A') returns 1
'A' is a printable character.
isprint(' ') returns 1
' ' is a printable character.
```

関連項目

ispunct

構文

#include <**ctype.h**>
int ispunct(int *c*);

目的

文字が句読点文字であるかどうかをテストします。

戻り値

ispunct 関数は、*c* が句読点文字ならばゼロ以外の値を返します。

パラメータ

c テストされる文字。

説明

ispunct 関数は、 **isspace** または **isalnum** のいずれもが true でないロケール固有の句読点文字セットの文字である任意の出力文字をテストします。“C”ロケールでは、 **ispunct** は、**isspace** または **isalnum** のいずれもが true でないすべての出力文字に対して true を返します。

例

```
/* a sample program testing ispunct() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c = 'A';
    printf("ispunct('%c') returns %d\n", c,ispunct(c));
    if(ispunct(c) > 0)
        printf("'%c' is a punctuation mark.\n", c);
    else
        printf("'%c' is not a punctuation mark.\n", c);
    c = ';';
    printf("ispunct('%c') returns %d\n", c,ispunct(c));
    if(ispunct(c) > 0)
        printf("'%c' is a punctuation mark.\n", c);
    else
        printf("'%c' is not a punctuation mark.\n",c);
}
```

出力

```
ispunct('A') returns 0
'A' is not a punctuation mark.
ispunct(';') returns 1
';' is a punctuation mark.
```

関連項目

isspace

構文

#include <**ctype.h**>
int isspace(int *c*);

目的

文字が空白文字であるかどうかをテストします。

戻り値

isspace 関数は、*c* が空白文字ならばゼロ以外の値を返します。

パラメータ

c テストされる文字。

説明

isspace 関数は、標準的な空白文字である、または isalnum が false であるロケール固有の文字セットの文字である任意の文字をテストします。標準的な空白文字を次に示します。スペース (**' '**)、改ページ (**'\f'**)、改行 (**'\n'**)、復帰 (**'\r'**)、水平タブ (**'\t'**)、および垂直タブ (**'\v'**)。"C"ロケールでは、 **isspace** は、標準的な空白文字に対してのみ true を返します。

例

```
/* a sample program testing isspace() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c = 'A';
    printf("isspace('%c') returns %d\n", c,isspace(c));
    if(isspace(c) > 0)
        printf("'%c' is a white space character.\n", c);
    else
        printf("'%c' is not a white space character.\n", c);
    c = ' ';
    printf("isspace('%c') returns %d\n", c,isspace(c));
    if(isspace(c) > 0)
        printf("'%c' is a white space character.\n", c);
    else
        printf("'%c' is not a white space character.\n",c);
}
```

出力

```
isspace('A') returns 0
'A' is not a white space character.
isspace(' ') returns 1
' ' is a white space character.
```

関連項目

isupper

構文

#include <**ctype.h**>
int isupper(**int** *c*);

目的

文字がアルファベット大文字であるかどうかをテストします。

戻り値

isupper 関数は、*c* がアルファベット大文字ならばゼロ以外の値を返します。

パラメータ

c テストされる文字。

説明

isupper 関数は、アルファベット大文字である、または **iscntrl** 、 **isdigit** 、 **ispunct** 、または **isspace** のいずれもが true でないロケール固有の文字セットの文字である任意の文字をテストします。“C”ロケールでは、 **isupper** は、アルファベット大文字であると定義された文字に対してのみ true を返します。

例

```
/* a sample program testing isupper() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c = 'A';
    printf("isupper('%c') returns %d\n", c,isupper(c));
    if(isupper(c) > 0)
        printf("'%c' is a upper case character.\n", c);
    else
        printf("'%c' is not a upper case character.\n", c);
    c = 'a';
    printf("isupper('%c') returns %d\n", c,isupper(c));
    if(isupper(c) > 0)
        printf("'%c' is a upper case character.\n", c);
    else
        printf("'%c' is not a upper case character.\n",c);
}
```

出力

```
isupper('A') returns 1
'A' is a upper case character.
isupper('a') returns 0
'a' is not a upper case character.
```

関連項目

isxdigit

構文

#include <**ctype.h**>
int isxdigit(int *c*);

目的

文字が 16 進数字であるかどうかをテストします。

戻り値

isxdigit 関数は、*c* が 16 進数字ならばゼロ以外の値を返します。

パラメータ

c テストされる文字。

説明

isxdigit 関数は、任意の 16 進数字をテストします。

例

```
/* a sample program testing isxdigit() */
#include <ctype.h>
#include <stdio.h>

int main() {
    char c = 'T';
    printf("isxdigit('%c') returns %d\n", c,isxdigit(c));
    if(isxdigit(c) > 0)
        printf("'%c' is a hexadecimal-digit character.\n", c);
    else
        printf("'%c' is not a hexadecimal-digit character.\n", c);
    c = 'a';
    printf("isxdigit('%c') returns %d\n", c,isxdigit(c));
    if(isxdigit(c) > 0)
        printf("'%c' is a hexadecimal-digit character.\n", c);
    else
        printf("'%c' is not a hexadecimal-digit character.\n",c);
}
```

出力

```
isxdigit('T') returns 0
'T' is not a hexadecimal-digit character.
isxdigit('a') returns 1
'a' is a hexadecimal-digit character.
```

関連項目

tolower

構文

#include <**ctype.h**>
int tolower(**int** *c*);

目的

アルファベット大文字を小文字に変換します。

戻り値

引数の **isupper** が true であり、現在のロケールによって指定された文字で **islower** が true である対応文字が 1 つ以上ある場合、**tolower** 関数は、対応文字のいずれかを返します (特定のロケールでは常に同じ文字を返します)。それ以外の場合、引数は変更されずに返されます。

パラメータ

c 変換される文字。

説明

tolower 関数は、アルファベット大文字を対応する小文字に変換します。

例

```
/* a sample program that displays lower case of the letter. */
#include <stdio.h>
#include <ctype.h>

int main() {
    char c = 'Q';
    printf("tolower('%c') = %c\n", c,tolower('Q'));
}
```

出力

```
tolower('Q') = q
```

関連項目

toupper

構文

#include <**ctype.h**>
int toupper(int *c*);

目的

アルファベット小文字を大文字に変換します。

戻り値

引数の **islower** が true であり、現在のロケールによって指定された文字で **isupper** が true である対応文字が 1 つ以上ある場合、 **toupper** 関数は、対応文字のいずれかを返します (特定のロケールでは常に同じ文字を返します)。それ以外の場合、引数は変更されずに返されます。

パラメータ

c 変換される文字。

説明

toupper 関数は、アルファベット小文字を対応する大文字に変換します。

例

```
/* a sample program that displays uppercase of the letter. */
#include <stdio.h>
#include <ctype.h>

int main() {
    char c = 'a';
    printf("toupper('%c') = %c\n", c, toupper(c));
}
```

出力

```
toupper('a') = A
```

関連項目

第6章　エラー — <errno.h>

ヘッダー **errno.h** には複数のマクロが定義されていて、すべてエラー状況の報告に関するマクロです。

定義されているマクロは次のとおりです。
EDOM
EILSEQ
ERANGE
これらのマクロは **int** 型の異なる正の値を持つ整数定数式に展開され、**#if** プリプロセッサディレクティブでの使用に適しています。また、

errno

は、 **int** 型の変更可能な lvalue に展開されます。lvalue には複数のライブラリ関数によって正のエラー番号が値として設定されます。
errno の値はプログラムの起動時はゼロですが、ライブラリ関数によってゼロに設定されることはありません。**errno** の値は、エラーの有無にかかわらず、ライブラリ関数の呼び出しによってゼロ以外に設定されることがあります。ただし、この国際標準の関数の説明で **errno** の使用についての記述がないことが前提です。

E と数字または **E** と大文字で始まる追加のマクロ定義が *errno.h* ヘッダーファイルで指定されます。

マクロ
errno.h ヘッダーファイルには次のマクロが定義されています。

マクロ	説明
EDOM	演算引数が関数の定義域外
EILSEQ	無効なバイトシーケンス
ERANGE	演算結果が表現できない

移植性
このヘッダーには移植性に関する既知の問題はありません。

第7章　浮動小数点型の特性 <float.h>

ヘッダー **float.h** には、標準の浮動小数点型のさまざまな制限とパラメータに展開される複数のマクロが定義されています。

マクロ名、マクロの意味、およびマクロの値の制約 (または制限) を以下に示します。

マクロ

float.h ヘッダーファイルには次のマクロが定義されています。

マクロ	値の制約	説明
FLT_RADIX	**2**	指数表現の基数

マクロ	値の制約	説明
FLT_MANT_DIG **DBL_MANT_DIG** **LDBL_MANT_DIG**	**24** **53**	**FLT_RADIX** を基数とする 浮動小数点の有効桁数

マクロ	値の制約	説明
FLT_DIG **DBL_DIG** **LDBL_DIG**	**6** **15**	小数点桁数

マクロ	値の制約	説明

FLT_MIN_EXP **DBL_MIN_EXP** **LDBL_MIN_EXP**	**(- 125)** **(- 1021)**	**FLT_RADIX** をその乗数よりも 1 小さい値で累乗すると正規化され る最小の負の整数

マクロ	値の制約	説明
FLT_MIN_10_EXP **DBL_MIN_10_EXP** **LDBL_MIN_10_EXP**	**(- 37)** **(- 307)**	10 をその乗数で累乗すると、 数字ではない範囲になる最小の 負の整数

マクロ	値の制約	説明
FLT_MAX_EXP **DBL_MAX_EXP** **LDBL_MAX_EXP**	**128** **1024**	**FLT_RADIX** をその乗数より小さい値で 累乗すると、表現可能な有限浮動小数点数となる 最大の整数

マクロ	値の制約	説明
FLT_MAX_10_EXP **DBL_MAX_10_EXP** **LDBL_MAX_10_EXP**	**38** **308**	10 をその乗数で累乗すると、 表現可能な有限浮動小数点数の 範囲になる最大の整数

マクロ	値の制約	説明
FLT_MAX **DBL_MAX** **LDBL_MAX**	**3.40282347E+38F** **1.797693134862315E+308**	表現可能な最大の有限浮動小数点数

マクロ	値の制約	説明
FLT_EPSILON **DBL_EPSILON** **LDBL_EPSILON**	**1.19209290E-07F** **2.2204460492503131E-16**	1 より大きい最小値と 1 との差であって、1 と最小値の間で表現可能な値

マクロ	値の制約	説明
FLT_MIN **DBL_MIN** **LDBL_MIN**	**1.17549435E-38F** **2.2250738585072014E-308**	正規化された最小の正の浮動小数点数

移植性

このヘッダーには移植性に関する既知の問題はありません。

第8章　整数型のサイズ — <limits.h>

ヘッダー **limits.h** では、標準の整数型のさまざまな制限とパラメータに展開される複数のマクロが定義されています。

マクロ名、マクロの意味、およびマクロの値の制約 (または制限) を以下に示します。

移植性

このヘッダーには移植性に関する既知の問題はありません。

マクロ

limits.h ヘッダーファイルには次のマクロが定義されています。

マクロ	説明 値の制約
CHAR_BIT	bit_filed(バイト) でない最小オブジェクトのビット数 **8**
SCHAR_MIN	**signed char** 型のオブジェクトの最小値 **-128** // -2^7
SCHAR_MAX	**signed char** 型のオブジェクトの最大値 **+127** // 2^7-1
UCHAR_MAX	**unsigned char** 型のオブジェクトの最大値 **255** // 2^8-1
CHAR_MIN	**char** 型のオブジェクトの最小値 **-128** // -2^7
CHAR_MAX	**char** 型のオブジェクトの最大値 **+127** // 2^7-1
MB_LEN_MAX	サポートされている任意のロケールに対する、マルチバイト文字の最大バイト数 **1** (この値はプラットフォームに依存します。)
SHRT_MIN	**short int** 型のオブジェクトの最小値 **-32768** // -2^{15}
SHRT_MAX	**short int** 型のオブジェクトの最大値 **+32767** // 2^{15}-1
USHRT_MAX	**unsigned short int** 型のオブジェクトの最大値 **65535** // 2^{16}-1
INT_MIN	**int** 型のオブジェクトの最小値

	-2147483648 // -2^{31}
INT_MAX	**int** 型のオブジェクトの最大値
	+2147483647 // $2^{31}-1$
UINT_MAX	**unsigned int** 型のオブジェクトの最大値
	4294967295 // $2^{32}-1$
LONG_MIN	**long int** 型のオブジェクトの最小値
	-2147483648 // -2^{31}
LONG_MAX	**long int** 型のオブジェクトの最大値
	2147483647 // $2^{31}-1$
ULONG_MAX	**unsigned long int** 型のオブジェクトの最大値
	4294967295 // $2^{32}-1$
LLONG_MIN	**long long int** 型のオブジェクトの最小値
	-9223372036854775808 // -2^{63}
LLONG_MAX	**long long int** 型のオブジェクトの最大値
	+9223372036854775807 // $2^{63}-1$
ULLONG_MAX	**unsigned long long int** 型のオブジェクトの最大値
	18446744073709551615 // $2^{64}-1$

第9章　ローカライズ <locale.h>

ヘッダー **locale.h** には 2 つの関数と 1 つの型が宣言されていて、複数のマクロが定義されています。宣言されている型は次の型です。

struct lconv

この型は数値の書式に関連したメンバを格納します。この構造体には次のメンバが任意の順序で格納されています。各メンバの意味と標準的な範囲については、後で説明します。Ch ロケールでは、各メンバは以下のコメントに示す値を持ちます。

```
char    *decimal_point;       //  "."
char    *thousands_sep;       //  ""
char    *grouping;            //  ""
char    *mon_decimal_point    //  ""
char    *mon_thousands_sep;   //  ""
char    *mon_grouping;        //  ""
char    *positive_sign;       //  ""
char    *negative_sign;       //  ""
char    *currency_symbol;     //  ""
char    frac_digits;          //  CHAR_MAX
 char   p_cs_precedes;        //  CHAR_MAX
char    n_cs_precedes;        //  CHAR_MAX
char    p_sep_by_space;       //  CHAR_MAX
char    n_sep_by_space;       //  CHAR_MAX
char    p_sign_posn;          //  CHAR_MAX
char    n_sign_posn;          //  CHAR_MAX
char    *int_curr_symbol;     //  ""
char    int_frac_digits;      //  CHAR_MAX
char    int_p_cs_precedes;    //  CHAR_MAX
char    int_n_cs_precedes;    //  CHAR_MAX
char    int_p_sep_by_space;   //  CHAR_MAX
char    int_n_sep_by_space;   //  CHAR_MAX
char    int_p_sign_posn;      //  CHAR_MAX
char    int_n_sign_posn;      //  CHAR_MAX
```

定義されているマクロは次のとおりです。

LC_ALL

LC_COLLATE

LC_CTYPE

LC_MONETARY

LC_NUMERIC

LC_TIME

これらのマクロは異なる値を持つ整数定数式に展開され、setlocale 関数の最初の引数としての使用に適しています。文字 **LC_**で始まる大文字のマクロ定義は今後のバージョンで追加される可能性があります。

関数

locale.h ヘッダーファイルには次の関数が定義されています。

関数	説明
localeconv()	現在のロケールに従った数量の書式として 適切な値を含むポインタを返します。
setlocale()	カテゴリによって指定されているプログラムの適切なロケールを選択します。プログラムの国際環境を変更または取得するために使用することもできます。

マクロ

locale.h ヘッダーファイルには次のマクロが定義されています。

マクロ	説明
LC_ALL	すべてのローカライズカテゴリを決定します。
LC_COLLATE	文字列照合関数である **strcoll()** と **strxfrm()** の動作に影響します。
LC_CTYPE	文字列分類関数と文字列変換関数の動作に 影響します。
LC_MONETARY	通貨単位の書式に影響します。
LC_NUMERIC	書式設定された入力/出力関数の小数点文字を 決定します。
LC_TIME	**strftime()** 関数の動作を決定します。

宣言される型

locale.h ヘッダーファイルによって、次の型が定義されます。

型	説明
lconv	struct - 数値の書式に関する値を保持します。

移植性
このヘッダーには移植性に関する既知の問題はありません。

localeconv

構文

#include <**locale.h**>
structlconv ∗**localeconv(void);**

目的

数値の書式設定規則を取得します。

戻り値

localeconv 関数は、設定済みオブジェクトへのポインタを返します。戻り値が参照する構造体をプログラムで修正することはできませんが、**localeconv** 関数への以降の呼び出しで上書きすることはできます。さらに、カテゴリが **LC_ALL**、**LC_MONETARY**、または **LC_NUMERIC** である **setlocale** 関数への呼び出しで、この構造体の内容を上書きできます。

パラメータ

引数はありません。

説明

localeconv 関数は、struct lconv 型のオブジェクトのメンバに、現在のロケールのルールに従った適切な数量の書式 (通貨など) を持つ値を設定します。
構造体の **char** ∗ 型のメンバは文字列へのポインタであり、**decimal_point** 以外のすべてのポインタは、""をポイントすることで、値が現在のロケールでは使用できないか、値の長さがゼロであることを示せます。**grouping** と **mon_grouping** を除き、これらの文字列は、初期シフト状態で始まり、初期シフト状態で終わる必要があります。**char** 型のメンバは非負数であり、 **CHAR_MAX** にすることで、値が現在のロケールでは使用できないことを示せます。次のメンバが含まれます。

char ∗**decimal_point**
金額以外の数量の書式を設定するために使用される小数点文字。

char ∗**thousands_sep**
書式設定された金額以外の数量において、小数点の前の数字の各グループを区切るために使用される文字。

char ∗**grouping**
書式設定された金額以外の数量において、その要素が数字の各グループのサイズを示す文字列。

char ∗**mon_decimal_point**
金額の書式を設定するために使用される小数点。

char ∗mon_thousands_sep
書式設定された金額において、小数点の前の数字の各グループを区切るために使用される区切り文字。

char ∗mon_grouping
書式設定された金額において、その要素が数字の各グループのサイズを示す文字列。

char ∗positive_sign
書式設定された負数ではない金額を示すために使用される文字列。

char ∗negative_sign
書式設定された負数の金額を示すために使用される文字列。

char ∗currency_symbol
現在のロケールに適用できる地域別通貨記号。

char frac_digits
ローカルに書式設定された金額で表示される小数点の後ろの数字の桁数。

char p_cs_precedes
ローカルに書式設定された負ではない金額の値に対して currency_symbol が先行する場合は 1 を、後続する場合は 0 を設定します。

char n_cs_precedes
ローカルに書式設定された負の金額の値に対して currency_symbol が先行する場合は 1 を、後続する場合は 0 を設定します。

char p_sep_by_space
currency_symbol、符号文字列、およびローカルに書式設定された負ではない金額の値の区切りを示す値を設定します。

char n_sep_by_space
currency_symbol、符号文字列、およびローカルに書式設定された負の金額の値の区切りを示す値を設定します。

char p_sign_posn
ローカルに書式設定された負ではない金額の **positive_sign** の位置を示す値を設定します。

char n_sign_posn
ローカルに書式設定された負の金額の **negative_sign** の位置を示す値を設定します。

char ∗int_curr_symbol
現在のロケールに適用できる国際通貨記号。最初の 3 文字は、ISO 4217:1995 の指定に従ったアルファ

ベットの国際通貨記号です。4 番目の文字 (null 文字の直前の文字) は、国際通貨記号と金額を区切るために使用される文字です。

char int_frac_digits
国際的に書式設定された金額で表示される小数点の後ろの数字の桁数。

char int_p_cs_precedes
国際的に書式設定された負ではない金額の値に対して **int_currency_symbol** が先行する場合は 1 を、後続する場合は 0 を設定します。(現行バージョンでは、まだ実装されていません。)

char int_n_cs_precedes
国際的に書式設定された負の金額の値に対して **int_currency_symbol** が先行する場合は 1 を、後続する場合は 0 を設定します。(現行バージョンでは、まだ実装されていません。)

char int_p_sep_by_space
int_currency_symbol、符号文字列、および国際的に書式設定された負ではない金額の値の区切りを示す値を設定します。(現行バージョンでは、まだ実装されていません。)

char int_n_sep_by_space
int_currency_symbol、符号文字列、および国際的に書式設定された負の金額の値の区切りを示す値を設定します。(現行バージョンでは、まだ実装されていません。)

char int_p_sign_posn
国際的に書式設定された負ではない金額の **positive_sign** の位置を示す値を設定します。(現行バージョンでは、まだ実装されていません。)

char int_n_sign_posn
国際的に書式設定された負の金額の **negative_sign** の位置を示す値を設定します。

grouping と **mon_grouping** の各要素は、次に従って解釈されます。

CHAR_MAX	これ以上のグループ化は実行されません。
0	直前の要素が残りの数字に対して繰り返し使用されます。
other	整数値は、現在のグループを構成する数字の桁数です。 現在のグループの前の次の数字のグループのサイズを決定するために、 次の要素が調べられます。

p_sep_by_space、**n_sep_by_space** の値は、次に従って解釈されます。

0 通貨記号と値を区切るスペースはありません。
1 通貨記号と符号文字列が隣接する場合は、それらと値をスペースで区切ります。
それ以外の場合は、通貨記号と値をスペースで区切ります。
2 通貨記号と符号文字列が隣接する場合は、それらをスペースで区切ります。
それ以外の場合は、符号文字列と値をスペースで区切ります。

p_sign_posn、**n_sign_posn** の各値は、次に従って解釈されます。

0 金額と通貨記号をかっこで囲みます。
1 金額と通貨記号の前に符号文字列を付けます。
2 金額と通貨記号の後ろに符号文字列を付けます。
3 通貨記号の直前に符号文字列を付けます。
4 通貨記号の直後に符号文字列を付けます。

実装では、ライブラリ関数による localeconv 関数の呼び出しが存在しないかのように動作する必要があります。

例 1 次の表は、4 か国で金額の書式を設定するためによく使用されるルールを示しています。

	ローカル書式		国際書式	
国	正	負	正	負
フィンランド	**1.234,56 mk**	**-1.234,56 mk**	**FIM 1.234,56**	**FIM -1.234,56**
イタリア	**L.1.234**	**-L.1.234**	**ITL 1.234**	**-ITL 1.234**
オランダ	**f 1.234,56**	**f -1.234,56**	**NLG 1.234,56**	**NLG -1.234,56**
スイス	**SFrs.1,234.56**	**SFrs.1,234.56**	**CHF 1,234.56**	**CHF -1,234.56**

これらの 4 か国では、localeconv によって返される構造体の通貨メンバの値は次のようになります。[1]

[1] int_p_cs_precedes、int_n_cs_precedes、int_p_sep_by_space、int_n_sep_by_space、int_p_sign_posn、int_n_sign_posn は、現行バージョンではまだ実装されていません。

	フィンランド	イタリア	オランダ	スイス
mon_decimal_point	**”,”**	**””**	**”,”**	**”.”**
mon_thousands_sep	**”.”**	**”.”**	**”.”**	**”,”**
mon_grouping	**”\3”**	**”\3”**	**”\3”**	**”\3”**
positive_sign	**””**	**””**	**””**	**””**
negative_sign	**”-”**	**”-”**	**”-”**	**”C”**
currency_symbol	**”mk”**	**”L.”**	**”\u0192”**	**”SFrs.”**
frac_digits	**2**	**0**	**2**	**0**
p_cs_precedes	**0**	**1**	**1**	**1**
n_cs_precedes	**0**	**1**	**1**	**1**
p_sep_by_space	**1**	**0**	**1**	**0**
n_sep_by_space	**1**	**0**	**1**	**0**
p_sign_posn	**1**	**1**	**1**	**1**
n_sign_posn	**1**	**1**	**4**	**2**
int_curr_symbol	**”FIM”**	**”ITL”**	**”NLG”**	**”CHF”**
int_facr_digits	**2**	**0**	**2**	**2**
int_p_cs_precedes	**1**	**1**	**1**	**1**
int_n_cs_precedes	**1**	**1**	**1**	**1**
int_p_sep_by_space	**0**	**0**	**0**	**0**
int_n_sep_by_space	**0**	**0**	**0**	**0**
int_p_sign_posn	**1**	**1**	**1**	**1**
int_n_sign_posn	**4**	**1**	**4**	**2**

例 2 次の表は、p_cs_precedes、p_sep_by_space、および p_sign_posn の各メンバが書式設定される値にどのように影響するかを示しています。

p_cs_precedes	p_sign_posn	p_sep_by_space		
		0	1	2
0	0	(1.25$)	(1.25 $)	(1.25$)
	1	+1.25$	+1.25 $	+ 1.25$
	2	1.25$+	1.25 $+	1.25 $ +
	3	1.25+$	1.25 +$	1.25 + $
	4	1.25$+	1.25 $+	1.25 $ +
1	0	(1.25)	($ 1.25)	($1.25)
	1	+$1.25	+$ 1.25	+ $1.25
	2	$1.25+	$ 1.25+	$1.25 +
	3	+$1.25	+$ 1.25	+ $1.25
	4	$+1.25	$+1.25	$ +1.25

例

```
/* a sample program that displays the decimal point
character used by the current locale. */
#include <stdio.h>
#include <locale.h>

int main() {
    struct lconv *lc;
    lc = localeconv();
    printf("Decimal symbol is: %s\n", lc->decimal_point);
    printf("Thousands separator is: %s\n", lc->thousands_sep);
    printf("Currency symbol is: %s\n", lc->currency_symbol);
}
```

出力

```
Decimal symbol is: .
Thousands separator is:
Currency symbol is:
```

関連項目

setlocale

構文

#include <**locale.h**>
char ∗**setlocale**(**int** *category*, **const char**∗ *locale*);

目的

ロケールを制御します。

戻り値

locale に対して文字列へのポインタが与えられ、選択を承認できる場合、 **setlocale** 関数は、新しいロケールのために指定された *category* に関連付けられた文字列へのポインタを返します。選択を承認できない場合、**setlocale** 関数は null ポインタを返し、プログラムのロケールは変更されません。

locale に対する null ポインタによって、**setlocale** 関数は、プログラムの現在のロケールの *category* に関連付けられた文字列へのポインタを返します。プログラムのロケールは変更されません。

setlocale 関数によって返される文字列へのポインタは、その文字列値および関連付けられたカテゴリを使用する以降の呼び出しで、プログラムのロケールのその部分が復元されるようにします。ポイントされた文字列をプログラムで修正してはなりませんが、**setlocale** 関数への以降の呼び出しで上書きすることはできます。

パラメータ

category ロケールのカテゴリ。.

locale ロケール値。

説明

setlocale 関数は、引数 category および locale によって指定されたプログラムのロケールの適切な部分を選択します。**setlocale** 関数を使用して、プログラム全体の現在のロケールまたはその一部を変更または取得できます。category に対する **LC_ALL** は、プログラム全体のロケールを指定します。category に対するそれ以外の値は、プログラムのロケールの一部のみを指定します。**LC_COLLATE** は、**strcoll** 関数と **strxfrm** 関数の動作に影響します。**LC_CTYPE** は、文字処理関数、マルチバイト関数、およびワイド文字関数の動作に影響します。**LC_MONETARY** は、**localeconv** 関数によって返される通貨の書式情報に影響します。**LC_NUMERIC** は、書式設定された入力/出力関数の小数点文字、文字列変換関数、および **localeconv** 関数によって返される通貨以外の書式情報に影響します。**LC_TIME** は、**strftime** 関数の動作に影響します。

locale に対する値”C”は、 $\mathbf{C}^{\mathbf{H}}$ 変換の最小限の環境を指定します。locale に対する値""は、ロケール固有のネイティブ環境を指定します。それ以外の実装によって定義された文字列は、**setlocale** への 2

番目の引数として渡すことができます。

プログラムの起動時に、

setlocale(LC_ALL, ”C”);

に相当する関数が実行されます。
実装では、ライブラリ関数による **setlocale** 関数の呼び出しが存在しないかのように動作する必要があります。

例

```
/* Sample program that demonstrates setlocale() function */
#include <stdio.h>
#include <locale.h>

int main()
{
    struct lconv* lc;

    printf("Current locale is : %s\n", setlocale(LC_ALL, NULL));
    lc = localeconv();
    printf("Example Amount : %d %s\n", 1000, lc->int_curr_symbol);
    printf("Change to local locale\n");
    setlocale(LC_ALL, "");
    printf("Current locale is : %s\n", setlocale(LC_ALL, NULL));
    lc = localeconv();
    printf("Example Amount : %d %s\n", 1000, lc->int_curr_symbol);
    return 0;
}
```

出力

```
Current locale is : C
Example Amount : 1000
Change to local locale
Current locale is : Japanese_Japan.932
Example Amount : 1000 JPY
```

関連項目

strcoll(), **strftime()**, **strxfrm()**.

第10章　算術 <math.h>

ヘッダー **math.h** では、2 種類のデータ型と複数の算術関数を宣言し、複数のマクロを定義します。ほとんどの構文には、1 つ以上の double 型パラメータ、または 1 つの double 型の戻り値、あるいはその両方を持つ主要関数と、同じ名前でサフィックス **f** または **l** が付加されたその他の関数で構成される関数ファミリが指定されています。f のサフィックス付きの関数は float 型のパラメータまたは戻り値 (あるいはその両方) を持ち、サフィックス l の関数は long double 型のパラメータまたは戻り値 (あるいはその両方) を持ちます。

次のマクロ

HUGE_VAL

は、正の double 型定数式に展開され、必ずしも **float** 型として表現できません。

次のマクロ

INFINITY

は、正または符号なしの無限大を表す float 型の定数式に展開されます。それができない環境では、変換時にオーバーフローする float 型の正の定数に展開されます。

次のマクロ

NAN

は、 **float** 型がシグナルなしの NaN をサポートしている実装でのみ定義されます。シグナルなしの **NaN** を表現できる **float** 型の定数式に展開されます。

次のマクロ

FP_ILOGB0

FP_ILOGBNAN

は、**ilogb(x)** 関数の戻り値に使用される整数定数式に展開されます。**x** がゼロの時は **FP_ILOGB0** 、**x** が NaN の時は **FP_ILOGBNAN** が返されます。**FP_ILOGB0** の値は **INT_MIN** または **INT_MAX** 、**FP_ILOGBNAN** の値は **INT_MAX** または **INT_MIN** です。

関数

math.h ヘッダーファイルには次の関数が定義されています。

関数	説明
acos()	引数のアークコサインを計算します。
acosh()	引数の逆双曲線コサインを計算します。
asin()	引数のアークサインを計算します。
asinh()	引数の逆双曲線サインを計算します。
atan()	引数のアークタンジェントを [-π/2, π/2] の範囲で計算します。
atan2()	引数のアークタンジェントを [-π, π] の範囲で計算します。
atanh()	引数の逆双曲線タンジェントを計算します。
cbrt()（*訳注）	引数の立方根を計算します。
ceil()	引数の値以上の最小の整数値を計算します。
copysign()	2 つの引数の符号を切り替えます。
cos()	引数のコサインを計算します。
cosh()	引数の双曲線コサインを計算します。
erf()（*訳注）	誤差関数を計算します。
erfc()（*訳注）	相補誤差関数を計算します。
exp()	*e* を底とする引数の指数関数を計算します。
exp2()	2 を底とする引数の指数関数を計算します。
expm1()	*e*e を底とする引数の指数関数-1 を計算します。
fabs()	引数の絶対値を計算します。
fdim()	2 つの引数の差を計算します。
floor()	引数の値以下の最大の整数値を計算します。
fma()	**(x+y)**×**z** を計算します。
fmax()	2 つの引数の最大値を計算します。
fmin()	2 つの引数の最小値を計算します。
fmod()	2 つの引数を除算した剰余を計算します。
frexp()	引数の正規化された小数部を計算します。
hypot()	$\sqrt{z^2 + y^2}$ を計算します
ilogb()（*訳注）	引数の指数を符号付き整数値として計算します。
ldexp()	**x**×2^{exp} を計算します。
lgamma()（*訳注）	引数の絶対対数ガンマ値を計算します。
log()	引数の対数を計算します。
log10()	引数の 10 を底とする対数を計算します。
log1p()	引数+1 の *e* を底とする対数を計算します。
log2()	引数の 2 を底とする対数を計算します。
logb()	引数の符号付き指数を計算します。
lrint()	現在の丸め方向に従って引数を整数値に丸めます。
lround()	引数を整数値に丸めます。値が中央にある場合は、 現在の丸め方向に関係なく、ゼロに丸めます。

modf()	引数の符号付き小数部を計算します。
nan()	シグナルなしの NaN を返します。
nearbyint()	現在の丸め方向に従って引数を整数値に丸めます。
nextafter()	引数の次に表現可能な値を計算します。
nexttoward()	引数の次に表現可能な値を計算します。
pow()	$\mathbf{x}^{\mathbf{y}}$ を計算します。
remainder()（*訳注）	2 つの引数を除算した剰余を計算します。
remquo()	2 つの引数を除算した剰余を計算します。
rint()（*訳注）	引数を整数値に丸めます。
round()	引数を整数値に丸めます。値が中央にある場合は、現在の丸め方向に関係なく、ゼロから遠い方に丸めます。
scalbn()（*訳注）	$\mathbf{x}\times\mathbf{FLT\_RADIX}^{n}$ を計算します。
sin()	引数のサインを計算します。
sinh()	引数の双曲線サインを計算します。
sqrt()	引数の平方根を計算します。
tan()	引数のタンジェントを計算します。
tanh()	引数の双曲線タンジェントを計算します。
tgamma()	引数のガンマ関数を計算します。
trunc()	引数を整数値に切り捨てます。

マクロ

math.h ヘッダーファイルには次のマクロが定義されています。

マクロ	説明
HUGE_VAL	double 型 － 正の定数式
INFINITY	float 型 － 正の無限大
NAN	float 型 － Not-a-Number(非数値)
isfinite	int 型 － 引数が有限の場合にゼロ以外の値を返します。
isgreater	double 型 － **x**>**y** の値を返します。
isgreaterequal	double 型 － **x**≥**y** の値を返します。
isinf	int - int 型 － 引数が無限大の場合にゼロ以外の値を返します。
isless	double 型 － **x**<**y** の値を返します。
islessequal	double 型 － **x**≤**y** の値を返します。
islessgreater	double 型 － **x** が **[y] より小さいか、[y] より大きいか、y** に等しいかを判断します。
isnan	int 型 － 引数が NaN である場合にゼロ以外の値を返し

	ます。
isnormal（*訳注）	int 型 ― 引数が正規化されている場合 (ゼロ、非正規化、無限大、NaN 以外の場合) にゼロ以外の値を返します。
isunordered	引数が順序付けられていない場合に 1 を返します。
signbit	int 型 ― 引数が負の値である場合にゼロ以外の値を返します。

移植性

isnormal() 、 **fpclassify**() 、 **signbit**() の各関数は、一部のプラットフォームでは使用できません。「（*訳注）」の注釈のある関数は、現行バージョンの Ch では、Windows プラットフォーム上でサポートされていません。

isunordered() 、 **lrint**() 、 **lround**() 、 **NAN**() 、 **nearbyint**() 、 **nexttoward**() 、**remquo**() 、 **round**() 、 **tgamma**() 、 **trunc**() の各関数は、どのプラットフォームでもサポートされていません。

acos

構文

#include <**math.h**>
type acos(**type** *x*);
double acos(**double** *x*);

戻り値

アークコサインを計算します。

戻り値

acos 関数は、アークコサインを [0, π] ラジアンの範囲内で返します。

パラメータ

x アークコサインを計算する値。

説明

acos 関数は *x* のアークコサインを計算します。
　引数が [-1, +1] の範囲外にある場合、領域エラーが発生します。この関数は、特殊な数字に関連する以下の値を返します。
— **acos**($\pm\infty$) は NaN を返します。
— **acos**(NaN) は NaN を返します。
— **acos**(± 0.0) は $\pi/2$ を返します。
— **acos**(x) は $|x| > 1$ の場合に NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("acos(x) = %f\n", acos(x));
}
```

出力

```
acos(x) = 1.266104
```

関連項目

cos(), **acosh**(), **cosh**().

acosh

構文

#include <**math.h**>
type acosh(**type** *x*);
double acosh(**double** *x*);

目的

逆双曲線コサインを計算します。

戻り値

acosh 関数は、逆双曲線コサインを [0,∞] の範囲内で返します。

パラメータ

x 逆双曲線コサインを計算する値。

説明

acosh 関数は *x* の負でない逆双曲線コサインを計算します。引数が 1 より小さい場合、領域エラーが発生します。この関数は、特殊な数字に関連する以下の値を返します。
— **acosh**($-\infty$) は NaN を返します。
— **acosh**($+\infty$) は $+\infty$ を返します。
— **acosh**(NaN) は NaN ををを返します。
— **acosh**($-x$) は $x > 0$ の場合に NaN を返します。
— **acosh**(x) は $x < 1.0$ の場合に NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("acosh(x) = %f\n", acosh(x));
}
```

出力

```
acosh(x) = NaN
```

関連項目

cos(), **acos**(), **cosh**().

asin

構文

#include <**math.h**>
type asin(**type** *x*);
double asin(**double** *x*);

目的

アークサインを計算します。

戻り値

asin 関数は、アークサインを [-π/2, π/2] の範囲内で返します。

パラメータ

x アークサインを計算する値。

説明

asin 関数は *x* のアークサインの主値を計算します。引数が [-1, +1] の範囲外にある場合、領域エラーが発生します。この関数は、特殊な数字に関連する以下の値を返します。
— **asin**($\pm\infty$) は NaN を返します。
— **asin(**NaN**)** は NaN を返します。
— **asin**(± 0.0) は ± 0.0 を返します。
— **asin**(x) は $|x| > 1$ の場合に NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("asin(x) = %f\n", asin(x));
}
```

出力

```
asin(x) = 0.304693
```

関連項目

sin(), **asinh**(), **sinh**().

asinh

構文

#include <**math.h**>
type asinh(**type** *x*);
double asinh(**double** *x*);

目的

逆双曲線サインを計算します。

戻り値

asinh 関数は逆双曲線サインの値を返します。この関数は、特殊な数字に関連する以下の値を返します。
— **asinh**($\pm\infty$) は $\pm\infty$ を返します。
— **asinh**(NaN) は NaN を返します。
— **asinh**(± 0.0) は ± 0.0 を返します。
— **asinh**($-x$) は $x > 0$ の場合に-**asin**(x) を返します。

パラメータ

x 逆双曲線サインを計算する値。

説明

asinh 関数は *x* の逆双曲線サインを計算します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("asinh(x) = %f\n", asinh(x));
}
```

出力

```
asinh(x) = 0.295673
```

関連項目

sin(), **asin**(), **sinh**().

atan

構文

#include <**math.h**>
type atan(**type** *x*);
double atan(**double** *x*);

目的

アークタンジェントを計算します。

戻り値

atan 関数は、アークタンジェントを [-π/2, π/2] ラジアンの範囲内で返します。

パラメータ

x アークタンジェントを計算する値。

説明

atan 関数は x のアークタンジェントを計算します。この関数は、特殊な数字に関連する以下の値を返します。
— **atan**($\pm\infty$) は $\pm\pi/2$ を返します。
— **atan**(± 0.0) は ± 0.0 を返します。
— **atan**(NaN) は NaN を返します。
— **atan**($-x$) は $x > 0$ の場合に $-$**atan**(x) を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("atan(x) = %f\n", atan(x));
}
```

出力

```
atan(x) = 0.291457
```

関連項目

tan(), **atanh**(), **atanh**().

atan2

構文

#include <**math.h**>
type atan2(**type** *y*, **type** *x*);
double atan2(**double** *y*, **double** *x*);

目的

アークタンジェントを計算します。

戻り値

atan2 関数は、**y/x** のアークタンジェントを [-π, π] ラジアンの範囲内で返します。

パラメータ

x アークタンジェントを計算する値の分母。

y アークタンジェントを計算する値の分子。

説明

atan2 関数は、*y/x* のアークタンジェントの主値を、両方の引数の符号を使用して計算し、戻り値の象限を求めます。両方の引数がゼロの場合、領域エラーが発生することがあります。この関数は、特殊な数字に関連する以下の値を返します。
— **atan2**($\pm 0, x$) は $x > 0$ の場合に ± 0 を返します。
— **atan2**($\pm 0, +0$) は ± 0 を返します。
— **atan2**($\pm 0, x$) は $x < 0$ の場合に $\pm \pi$ を返します。
— **atan2**($\pm 0, -0$) は $\pm \pi$ を返します。
— **atan2**($y, \pm 0$) は $y > 0$ の場合に $\pi/2$ を返します。
— **atan2**($y, \pm 0$) は $y < 0$ の場合に $\pi/2$ を返します。
— **atan2**($\pm y, \infty$) は $y \neq 0$ の場合に ± 0 を返します。
— **atan2**($\pm \infty, x$) は x が有限の場合に $\pm \pi/2$ を返します。
— **atan2**($\pm y, -\infty$) は $y > 0$ の場合に $\pm \pi$ を返します。
— **atan2**($\pm \infty, \infty$) は $\pm \pi/4$ を返します。
— **atan2**($\pm \infty, -\infty$) は $\pm 3\pi/4$ を返します。
— **atan2**(NaN, NaN) は NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    double y = 0.5;
    printf("atan2(x,y) = %f\n", atan2(x,y));
}
```

出力

```
atan2(x,y) = 0.540420
```

関連項目

tan(), **atan2h**(), **tan2h**().

atanh

構文

#include <**math.h**>
type atanh(**type** *x*);
double atanh(**double** *x*);

目的

逆双曲線タンジェントを計算します。

戻り値

atanh 関数は逆双曲線タンジェントの値を返します。

パラメータ

x 逆双曲線タンジェントを計算する値。

説明

atanh 関数は *x* の逆双曲線タンジェントを計算します。引数が [-1,1] の範囲外にある場合、領域エラーが発生します。引数が-1 または+1 に等しい場合、範囲エラーが発生することがあります。この関数は、特殊な数字に関連する以下の値を返します。
— **atanh**(±0) は ±0 を返します。
— **atanh**(±1) は $\pm\infty$ を返します。
— **atanh**(*x*) は $|x| > 1$ の場合に NaN を返します。
— **atanh**($\pm\infty$) は NaN を返します。
— **atanh**(NaN) は NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("atanh(x) = %f\n", atanh(x));
}
```

出力

```
atanh(x) = 0.309520
```

関連項目

tan(), **atan**(), **tanh**().

cbrt

構文

#include <**math.h**>
double cbrt(double *x*);

目的

立方根を計算します[1]。

戻り値

cbrt 関数は立方根の値を返します。

パラメータ

x 立方根の値を計算する値。

説明

cbrt 関数は *x* の実立方根を計算します。この関数は、特殊な数字に関連する以下の値を返します。
—**cbrt**(±∞) は ±∞ を返します。
—**cbrt**(±0) は ±0 を返します。
—**cbrt**(NaN) は NaN を返します。

例

```
/* a sample program that solves the cube root */
#include <stdio.h>
#include <math.h>

int main() {
    double x= 8.0;
    printf( "the cube root of 8 is %f\n",cbrt(x));
}
```

出力

```
the cube root of 8 is 2.000000
```

関連項目

[1]訳注：現行バージョンの Ch では、**cbrt** 関数は Windows プラットフォーム上でサポートされていません。

ceil

構文

#include <**math.h**>
type ceil(**type** *x*);
double ceil(**double** *x*);

目的

最小整数を計算します。

戻り値

ceil 関数は *x* 以上の最小整数値を浮動小数点数表現で返します。この関数は、特殊な数字に関連する以下の値を返します。
— **ceil**(±∞) は ±∞ を返します。
— **ceil**(±0) は ±0 を返します。
— **ceil**(NaN) は NaN を返します。

パラメータ

x それ以上の最小の整数値を計算する値。

説明

ceil 関数は、*x* 以上の最小の整数値を計算します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("ceil(x) = %f\n", ceil(x));
}
```

出力

```
ceil(x) = 1.000000
```

関連項目

copysign

構文

#include <**math.h**>
double copysign(double *x*, **double** *y*);

目的

値の符号を切り替えます。

戻り値

copysign 関数は、*x* の絶対値と *y* の符号を持った値を返します。

パラメータ

x 戻り値の絶対値。

y 戻り値の符号。

説明

copysign 関数は、*x* の絶対値と *y* の符号を持った値を返します。*x* が NaN の場合、NaN に *y* の符号を付けて返します。符号付きゼロの表現を実装していても、算術演算で負のゼロが一貫して処理されない場合は、**copysign** 関数はゼロの符号を正と見なします。

例

```
/* a sample program that changes the first number's
sign to the second number's */
#include <stdio.h>
#include <math.h>

int main() {
    double x,y,z;

    x = -1.563;
    y = 3.7664;
    printf("before copysign x = %f,  y = %f \n",x,y);
    z = copysign(x,y);
    printf("after copysign x = %f, y = %f , z = %f\n", x, y, z);
}
```

出力

```
before copysign x = -1.566300, y = 3.766400
after copysign x = -1.566300, y = 3.766400 , z = 1.566300
```

関連項目

cos

構文

#include <**math.h**>
type cos(**type** *x*);
double cos(**double** *x*);

目的

コサインを計算します。

戻り値

cos 関数はコサインの値を返します。

パラメータ

x コサインを計算する値。

説明

cos 関数は、ラジアンで測定された *x* のコサインを計算します。この関数は、特殊な数字に関連する以下の値を返します。
— **cos**(±0) は 1 を返します。
— **cos**(±∞) は NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("cos(x) = %f\n", cos(x));
}
```

出力

```
cos(x) = 0.955336
```

関連項目

acos(), **acosh**(), **cosh**().

cosh

構文

#include <**math.h**>
type cosh(**type** *x*);
double cosh(**double** *x*);

目的

双曲線コサインを計算します。

戻り値

cosh 関数は双曲線コサインの値を返します。

パラメータ

x 双曲線コサインを計算する値。

説明

cosh 関数は *x* の双曲線コサインを計算します。*x* の絶対値が大きすぎる場合、範囲エラーが発生します。この関数は、特殊な数字に関連する以下の値を返します。
— cosh(±0) は 1 を返します。
— cosh(±∞) は ±∞ を返します。
— cosh(NaN) は NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
   double x = 0.3;
   printf("cosh(x) = %f\n", cosh(x));
}
```

出力

```
cosh(x) = 1.045339
```

関連項目

cos(), **acosh**(), **acos**().

erf

構文

#include <**math.h**>
double erf(double *x*);

目的

誤差関数を計算します[2]。

戻り値

erf 関数は誤差関数の値を返します。

パラメータ

x 誤差関数を計算する値。

説明

erf 関数は、x の誤差関数 $\frac{2}{\sqrt{\pi}} \int_0^{\mathbf{x}} e^{-t^2} dt$ を計算します。この関数は、特殊な数字に関連する以下の値を返します。
— **erf(**± 0**)** は ± 0 を返します。
— **erf(**$\pm \infty$**)** は ± 1 を返します。

例

```
/* a sample program that returns the error fuction of x */
#include <stdio.h>
#include <math.h>

int main() {
    double x,y;

    x = 1;
    y = erf(x);
    printf("the error fuction of x= %f is %f\n",x,y);
    y = erfc(x);
    printf("the complementary value of x= %f is %f\n",x,y);
}
```

出力

```
the error fuction of x= 1.000000 is 0.842701
the complementary value of x= 1.000000 is 0.157299
```

関連項目

[2]訳注：現行バージョンの Ch では、**erf** 関数は Windows プラットフォーム上でサポートされていません。

erfc

構文
#include <**math.h**>
double erfc(**double** *x*);

目的
相補誤差関数を計算します[3]。

戻り値
erfc 関数は、相補誤差関数の値を返します。

パラメータ
x 相補誤差関数を計算する値。

説明
erfc 関数は、*x* の相補誤差関数 $\frac{2}{\sqrt{\pi}}\int_{\mathbf{x}}^{\infty} e^{-t^2}\,dt$ を計算します。*x* の値が大きすぎる場合、範囲エラーが発生します。この関数は、特殊な数字に関連する以下の値を返します。
— **erfc**($+\infty$) は 0 を返します。
— **erfc**($-\infty$) は 2 を返します。

例
erf() を参照。

関連項目

[3]訳注：現行バージョンの Ch では、**erfc** 関数は Windows プラットフォーム上でサポートされていません。

exp

構文

#include <**math.h**>
type exp(**type** *x*);
double exp(**double** *x*);

目的

e を底とする指数関数を計算します。

戻り値

exp 関数は指数関数の値を返します。

パラメータ

x *e* を底とする指数関数の値を計算する値。

説明

exp 関数は、*e* を底とする *x* の指数関数 e^x を計算します。*x* の絶対値が大きすぎる場合、範囲エラーが発生します。この関数は、特殊な数字に関連する以下の値を返します。
— **exp**(± 0) は 1 を返します。
— **exp**($+\infty$) は $+\infty$ を返します。
— **exp**($-\infty$) は $+0$ を返します。
— **exp**(NaN) は NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("exp(x) = %f\n", exp(x));
}
```

出力

```
exp(x) = 1.349859
```

関連項目

exp2(), **expm1**(), **frexp**().

exp2

構文

#include <**math.h**>
double exp2(**double** *x*);
float exp2f(**float** *x*);
long double exp2l(**long double** *x*);

目的

2 を底とする指数関数を計算します。

戻り値

exp2 関数は 2 を底とする指数関数の値を返します。

パラメータ

x 2 を底とする指数関数の値を計算する値。

説明

exp2 関数は、2 を底とする *x* の指数関数 2^x を計算します。x の絶対値が大きすぎる場合、範囲エラーが発生します。この関数は、特殊な数字に関連する以下の値を返します。
— **exp2**(± 0) は 1 を返します。
— **exp2**($+\infty$) は $+\infty$ を返します。
— **exp2**($-\infty$) は $+0$ を返します。
— **exp2**(NaN) は NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main() {
    double x = 3.0;
    printf("exp2(%f) = %f\n", x, exp2(x));
}
```

出力

```
exp2(3.000000) = 8.000000
```

関連項目

exp(), **expm1**(), **frexp**().

expm1

構文

#include <**math.h**>
double expm1(**double** *x*);

目的

e を底とする指数関数-1 を計算します。

戻り値

expm1 関数は、$e^x - 1$ の値を返します。

パラメータ

x $e^x - 1$ の値を計算する値。

説明

expm1 関数は、*e* を底とする引数の指数関数から 1 を引いた値 $e^x - 1$ を計算します。x の値が大きすぎる場合、範囲エラーが発生します。この関数は、特殊な数字に関連する以下の値を返します。
— **expm1**(± 0) は 0 を返します。
— **expm1**($+\infty$) は $+\infty$ を返します。
— **expm1**($-\infty$) は -1 を返します。
— **expm1**(NaN) は NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("expm1(x) = %f\n", expm1(x));
}
```

出力

```
expm1(x) = 0.349859
```

関連項目

exp(), **exp2**(), **frexp**().

fabs

構文

#include <**math.h**>
double fabs(**double** *x*);

目的

絶対値を計算します。

戻り値

fabs 関数は x の絶対値を返します。

パラメータ

x 絶対値を計算する値。

説明

fabs 関数は、浮動小数点数 *x* の絶対値を計算します。この関数は、特殊な数字に関連する以下の値を返します。
— **fabs**(±0) は +0 を返します。
— **fabs**(±∞) は +∞ を返します。
— **fabs**(NaN) は NaN を返します。

例

```
/* a sample program that returns the absolute
value of x. */
#include <stdio.h>
#include <math.h>

int main() {
    float x;

    x = -12.5;
    printf("The absolute of %3.1f is %3.1f \n",x,fabs(x));
}
```

出力

```
The absolute of -12.5 is 12.5
```

関連項目

fdim

構文

#include <**math.h**>
double fdim(double *x*, **double** *y*);
float fdimf(float *x*, **float** *y*);
long double fdiml(long double *x*, **long double** *y*);

目的

差の値を計算します。

戻り値

fdim 関数は正の差の値を返します。

パラメータ

x 減算される値 (*y* よりも大きくなければなりません)。

y 減算する値 (*x* 以下でなければなりません)。

説明

fdim 関数は、引数間の正の差を求めます。
$x > y$ の場合は $x - y$
$x \leq y$ の場合は+0
範囲エラーが発生することがあります。

例

```
#include <stdio.h>
#include <math.h>

int main() {
    double x[2] = {1.0, 3.0};
    double y[2] = {3.0, 1.0};
    printf("fdim(x[0],y[0]) = %f\n", fdim(x[0], y[0]));
    printf("fdim(x[1],y[1]) = %f\n", fdim(x[1], y[1]));
}
```

出力

```
fdim(x[0],y[0]) = 0.000000
fdim(x[1],y[1]) = 2.000000
```

関連項目

floor

構文

#include <math.h>
type floor(type *x***);**
double floor(double *x***);**

目的

最大整数を計算します。

戻り値

floor 関数は *x* 以下の最大整数値を浮動小数点数表現で返します。

パラメータ

x それより小さい最大の整数値を計算する値。

説明

floor 関数は *x* 以下の最大整数値を計算します。この関数は、特殊な数字に関連する以下の値を返します。
— **floor**(*x*) は x が $\pm\infty$ もしくは ± 0 の場合、x を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("floor(x) = %f\n", floor(x));
}
```

出力

```
floor(x) = 0.000000
```

関連項目

fma

構文

#include <math.h>
double fma(double *x*, **double** *y*, **double** *z*);

目的

和と積を計算します。

戻り値

fma 関数は、1 つの 3 項演算として **(x+y)×z** を計算した後、その結果を丸めます。

パラメータ

x 和を計算する一方の値。

y 和を計算する他方の値。

z **x+y** に乗算される値。

説明

fma 関数は、1 つの 3 項演算として **(x+y)×z** を計算した後、その結果を丸めます。計算は値の精度が無限大であるものとして行い、**FLT_ROUNDS** の値で指定されている丸めモードに従って 1 回で所定の結果形式に丸めます。この関数は、特殊な数字に関連する以下の値を返します。
— x と y の一方が無限大で、他方がゼロの場合、**fma(**x, y, z**)** は NaN を返します。
— x と y の一方が明確な無限大であり、z も無限大だが符号は逆の場合、**fma(**x, y, z**)** は NaN を返します。

例

```
#include <stdio.h>
#include <math.h>

int main() {
    double x = 1.0, y = 2.0, z = 3.0;
    printf("fma(%f,%f,%f) = %f\n", x, y, z, fma(x, y, z));
}
```

出力

```
fma(1.000000,2.000000,3.000000) = 5.000000
```

関連項目

fmax

構文

#include <**math.h**>
double fmax(double *x*, **double** *y*);

目的

最大数値を計算します。

戻り値

fmax 関数は引数の最大数値を返します。

パラメータ

x 比較される一方の値。

y 比較される他方の値。

説明

fmax 関数は、引数の最大数値を求めます。この関数は、特殊な数字に関連する以下の値を返します。— 一方の引数が NaN の場合、 **fmax** はもう一方の引数を返します。引数がどちらも NaN の場合、NaN を返します。

例

```
/* a sample program that decides which of the two values is a max. */
#include <math.h>
#include <stdio.h>

int main() {
    double x, y, z;

    x = 1.0;
    y = 2.0;
    z =fmax(x, y);
    printf("Value 1: %f\n", x);
    printf("Value 2: %f\n", y);
    printf("%f is the maximum numeric value.\n",z);
}
```

出力

```
Value 1: 1.000000
Value 2: 2.000000
2.000000 is the maximum numeric value.
```

関連項目

fmin

構文

#include <**math.h**>
double fmin(double *x,* **double** *y*);

目的

最小数値を計算します。

戻り値

fmin 関数は引数の最小数値を返します。

パラメータ

x 比較される一方の値。

y 比較される他方の値。

説明

fmin 関数は、引数の最小数値を求めます。この関数は、特殊な数字に関連する以下の値を返します。
— 一方の引数が NaN の場合、 **fmin** はもう一方の引数を返します。引数がどちらも NaN の場合、NaN を返します。

例

```
/* a sample program that decides which of the two values is a min. */
#include <math.h>
#include <stdio.h>

int main() {
    double x, y, z;

    x = 1.0;
    y = 2.0;
    z =fmin(x, y);
    printf("Value 1: %f\n", x);
    printf("Value 2: %f\n", y);
    printf("%f is the minimum numeric value.\n",z);
}
```

出力

```
Value 1: 1.000000
Value 2: 2.000000
1.000000 is the minimum numeric value.
```

関連項目

fmod

構文

#include <**math.h**>
type fmod(type *x*, **double** *y*);
double fmod(double *x*, **double** *y*);
float fmodf(float *x*, **float** *y*);

目的

剰余を計算します。

戻り値

fmod 関数は *x*/*y* の浮動小数点数の剰余を計算します。

パラメータ

x 被除数。

y 除数。

説明

fmod 関数は、ある整数 *n* に対して *x*-*ny* の値を返します。y がゼロ以外の場合、結果は x と同じ符号を持ち、その絶対値は y の絶対値より小さくなります。y がゼロの場合、領域エラーが発生するか、ゼロを返すかを実装で定義します。この関数は、特殊な数字に関連する以下の値を返します。
— y がゼロ以外の場合、**fmod**($\pm 0, y$) は ± 0 を返します。
— x が無限大で y がゼロの場合、**fmod**(x, y) は NaN を返します。
— x が無限大ではない場合、**fmod**($x, \pm\infty$) は x を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    double y = 1.7;
    printf("fmod(x,y) = %f\n", fmod(x,y));
}
```

出力

```
fmod(x,y) = 0.300000
```

関連項目

fpclassify

構文
#include <**math.h**>
int fpclassify(double *x*);

目的
引数を分類します[4]。

戻り値
fpclassify マクロは、引数の値に適した数値分類マクロの値を返します。

パラメータ
x 分類される値。

説明
fpclassify マクロは、引数の値を NaN、無限大、正規化されている値、正規化されていない値、ゼロに分類します。引数が構文の型よりも幅が広い場合はその構文の型に変換してから、引数の種類に基づいて分類します。

関連項目

[4]訳注：現行バージョンの Ch では、**fpclassify** マクロは Windows プラットフォーム上でサポートされていません。

frexp

構文

#include <**math.h**>
double frexp(double *value*, **int** **exp*);

目的

数値を正規化された小数に変換します。

戻り値

frexp 関数は、絶対値が [1/2, 1] の範囲内またはゼロで与えられた *value* が 2 の ∗**exp** 乗の *x* 倍に等しくなる値 *x* を返します。*value* がゼロの場合、結果の指数部も仮数部もゼロです。

パラメータ

value 正規化される値。

exp 指数部を格納する整数型変数へのポインタ。

説明

frexp 関数は、浮動小数点数を正規化された小数部と 2 の累乗の整数部に分けます。*exp* が参照する **int** オブジェクトに整数部を格納します。この関数は、特殊な数字に関連する以下の値を返します。
— **frexp**($\pm 0, exp$) は ± 0 を返します。
— **frexp**($\pm\infty, exp$) は $\pm\infty$ を返します。
— **frexp**(NaN, *exp*) は NaN を返します。

例

```
#include <stdio.h>
#include <math.h>

int main() {
    double value = 3.1415926;
    double x;
    int expt;
    x = frexp(value, &expt);
    printf("x = %f\n", x);
    printf("*exp = %d\n", expt);
}
```

出力

```
x = 0.785398
*exp = 2
```

関連項目

exp2(), **expm1**(), **exp**().

hypot

構文

#include <**math.h**>
double hypot(double *x*, **double** *y*);

目的

平方和を計算します。

戻り値

hypot 関数は平方和の平方根の値を返します。

パラメータ

x 平方を計算する一方の値。

y 平方を計算する他方の値。

説明

hypot 関数は、不要なオーバーフローやアンダーフローを発生させないで、*x* の 2 乗と *y* の 2 乗の合計値の平方根を計算します。範囲エラーが発生することがあります。この関数は、特殊な数字に関連する以下の値を返します。
— x が無限の場合、y が NaN でも、**hypot**(x, y) は $\pm\infty$ を返します。
— **hypot**($x, \pm 0$) は **fabs**(x) と同等です。

例

```
/* a sample program that use hypot()*/
#include <stdlib.h>
#include <math.h>

int main() {
    double z,y;
    double x;

    x = 3;
    y = 4;
    z = hypot(x,y);
    printf("The magnitude of the complex number %2.1f+i%2.1f is %2.1f\n",x,y,z);
}
```

出力

```
 The magnitude of the complex number 3.0+4.0i is 5.0
```

関連項目

ilogb

構文

#include <**math.h**>
int ilogb(**double** *x*);

目的

指数を計算します[5]。

戻り値

ilogb 関数は、*x* の指数部を符号付き整数値として返します。

パラメータ

x その指数部を計算する値。

説明

ilogb 関数は、*x* の指数部を符号付き整数値として抽出します。*x* がゼロの場合、値 **FP_ILOGB0** を返します。*x* が無限大の場合、値 **INT_MAX** を返します。*x* が NaN の場合、値 **FP_ILOGBNAN** を返します。それ以外の場合、対応する **logb** 関数を呼び出して、その戻り値を **int** 型にキャストした結果を返します。*x* が 0 の場合、範囲エラーが発生することがあります。

例

```
/*  The ilogb functions return the signed exponent of x as a signed int value */
#include <stdlib.h>
#include <math.h>

int main() {
    double x = 10.0;
    printf("The result is %d\n",ilogb(x));
}
```

出力

```
The result is 3
```

関連項目

[5]訳注：現行バージョンの Ch では、**ilogb** 関数は Windows プラットフォーム上でサポートされていません。

isfinite

構文

#include <**math.h**>
int isfinite(double_t *x*);

目的

引数が有限値であるかどうかを判断します。

戻り値

isfinite マクロは、引数が有限の値の場合のみ、ゼロ以外の値を返します。

パラメータ

x 有限値であるかどうかをテストされる値。

説明

isfinite マクロは、引数が有限の値 (ゼロ、正規化されていない値、または正規化された値のいずれかであり、無限大でも NaN でもない値) かどうかを判断します。引数が構文の型よりも幅が広い場合はその構文の型に変換してから、引数の種類に基づいて判断します。

例

```
/* a sample program that decides if x is finite. */
#include <math.h>
#include <stdio.h>

int main() {
    double x;

    x = 1.0;
    if(isfinite(x))
        printf("%f is a finite number\n",x);
    else
        printf("%f is not a finite value\n", x);
}
```

出力

```
1.000000 is a finite number
```

関連項目

isgreater

構文

#include <**math.h**>
double isgreater(*real-floating x, real-floating y*);

目的

より大きい値を返します。

戻り値

isgreater マクロは **(***x***)** > **(***y***)** の値を返します。

パラメータ

x 比較される一方の値。

y 比較される他方の値。

説明

isgreater マクロは、1 番目の引数が 2 番目の引数より大きいかどうかを判断します。**isgreater**(*x*,*y*) の値は常に **(***x***)**>**(***y***)** に等しくなります。

例

```
#include <stdio.h>
#include <math.h>

int main() {
   double x = 2.0, y=1.0;
   printf("isgreater(%f %f)= %d\n", x, y, isgreater(x, y));
}
```

出力

```
isgreater(2.000000 1.000000)= 1
```

関連項目

isgreaterequal

構文

#include <**math.h**>
double isgreaterequal(*real-floating x, real-floating y*);

Purpose

ある値以上の値を返します。

戻り値

isgreaterequal マクロは **(***x***)** ≥ **(***y***)** の値を返します。

パラメータ

x 比較される一方の値。

y 比較される他方の値。

説明

isgreaterequal マクロは、1 番目の引数が 2 番目の引数以上であるかどうかを判断します。**isgreaterequal(***x***,***y***)** の値は常に **(***x***)** ≥ **(***y***)** に等しくなります。

例

```
#include <stdio.h>
#include <math.h>

int main() {
   double x = 2.0, y=1.0, z = 2.0;
   printf("isgreaterequal(%f %f)= %d\n", x, y, isgreaterequal(x, y));
   printf("isgreaterequal(%f %f)= %d\n", x, z, isgreaterequal(x, z));
}
```

出力

```
isgreaterequal(2.000000 1.000000)= 1
isgreaterequal(2.000000 2.000000)= 1
```

関連項目

isinf

構文

#include <**math.h**>
int isinf(double *x*);

目的

引数が無限大の値であるかどうかを判断します。

戻り値

isinf マクロは、引数が無限大の値の場合のみ、ゼロ以外の値を返します。

パラメータ

x 無限大であるかどうかをテストされる値。

説明

isinf マクロは、引数が無限大 (正または負) であるかどうかを判断します。引数が引数が構文の型よりも幅が広い場合はその構文の型に変換してから、引数の種類に基づいて判断します。

例

```
/* a sample program that decides if x is infinite. */
#include <math.h>
#include <stdio.h>

int main() {
    double x;

    x = 1.0;
    if(isinf(x))
        printf("%f is infinite.\n",x);
    else
        printf("%f is not infinite. \n", x);
    x = +INFINITY;
    if(isinf(x))
        printf("%f is infinite.\n",x);
    else
        printf("%f is not infinite. \n", x);
}
```

出力

```
1.000000 is not infinite.
Inf is infinite.
```

関連項目

isless

構文

#include <**math.h**>
double isless(*real-floating x, real-floating y*);

目的

より小さい値を返します。

戻り値

isless マクロは **(***x***)** < **(***y***)** の値を返します。

パラメータ

x 比較される一方の値。

y 比較される他方の値。

説明

isless マクロは、1 番目の引数が 2 番目の引数より小さいかどうかを判断します。**isless**(*x*,*y*) の値は常に **(***x***)**<**(***y***)** に等しくなります。

例

```
#include <stdio.h>
#include <math.h>

int main() {
   double x = 2.0, y=1.0;
   printf("isless(%f %f)= %d\n", x, y, isless(x, y));
}
```

出力

```
isless(2.000000 1.000000)= 0
```

関連項目

islessequal

構文

#include <**math.h**>
double islessequal(*real-floating x*, *real-floating y*);

目的

より小さい値を返します。

戻り値

islessequal マクロは、**(***x***)** ≤ **(***y***)** の値を返します。

パラメータ

x 比較される一方の値。

y 比較される他方の値。

説明

islessequal マクロは、1 番目の引数が 2 番目の引数以下であるかどうかを判断します。**islessequal(***x***,***y***)** の値は常に **(***x***)** ≤ **(***y***)** に等しくなります。

例

```
#include <stdio.h>
#include <math.h>

int main() {
   double x = 2.0, y=1.0, z = 2.0;
   printf("islessequal(%f %f)= %d\n", x, y, islessequal(x, y));
   printf("islessequal(%f %f)= %d\n", x, z, islessequal(x, z));
}
```

出力

```
islessequal(2.000000 1.000000)= 0
islessequal(2.000000 2.000000)= 1
```

関連項目

islessgreater

構文

#include <**math.h**>
double islessgreater(*real-floating x, real-floating y*);

目的

このマクロはゼロでない値を常に返します。

戻り値

islessgreater マクロは、**(***x***)** ≤ **(***y***)** || **(***x***)** ≥ **(***y***)** の値を返します。

パラメータ

x 比較される一方の値。

y 比較される他方の値。

説明

islessgreater マクロは、1 番目の引数が 2 番目の引数より小さいか、または大きいかどうかを判断します。**islessgreater** マクロは **(***x***)** ≤ **(***y***)** || **(***x***)** ≥ **(***y***)** と同様です。

例

```
#include <stdio.h>
#include <math.h>

int main() {
   double x = 2.0, y=1.0, z = 2.0;
   printf("islessgreater(%f %f)= %d\n", x, y, islessgreater(x, y));
   printf("islessgreater(%f %f)= %d\n", y, z, islessgreater(y, z));
}
```

出力

```
islessgreater(2.000000 1.000000)= 1
islessgreater(1.000000 2.000000)= 1
```

関連項目

isnan

構文

#include <**math.h**>
int isnan(*real-floating x*);

目的

引数の値が NaN であるかどうかを判断します。

戻り値

isnan マクロは、引数の値が NaN の場合のみ、ゼロ以外の値を返します。

パラメータ

x NaN であるかどうかをテストされる値。

説明

isnan マクロは、引数が NaN かどうかを判断します。引数が引数が構文の型よりも幅が広い場合はその構文の型に変換してから、引数の種類に基づいて判断します。

例

```
/* a sample program that assigns NaN (not a number) to x
and decide if it is NaN. */
#include <math.h>
#include <stdio.h>

int main() {
    double x;

    x = NaN;
    if(isnan(x)) printf("%f is not a number\n",x);
}
```

出力

```
NaN is not a number
```

関連項目

isnormal

構文

#include <**math.h**>
int isnormal(double *x*);

目的

引数が正規化された値であるかどうかを判断します[6]。

戻り値

isnormal マクロは、引数が正規化された値である場合のみ、ゼロ以外の値を返します。

パラメータ

x 正規化されているかどうかをテストされる値。

説明

isnormal マクロは、引数が正規化された値 (ゼロ、正規化されていない値、無限大、NaN 以外の値) であるかどうかを判断します。引数が引数が構文の型よりも幅が広い場合はその構文の型に変換してから、引数の種類に基づいて判断します。

例

```
#include <stdio.h>
#include <math.h>

int main() {
    printf("isnormal(3) = %d\n",  isnormal(3));
    printf("isnormal(-3) = %d\n",  isnormal(-3));
    printf("isnormal(0) = %d\n",  isnormal(0));
    printf("isnormal(-0) = %d\n",  isnormal(-0));
    printf("isnormal(Inf) = %d\n",  isnormal(Inf));
    printf("isnormal(-Inf) = %d\n",  isnormal(-Inf));
    printf("isnormal(NaN) = %d\n",  isnormal(NaN));
}
```

出力

```
isnormal(3) = 1
isnormal(-3) = 1
isnormal(0) = 0
isnormal(-0) = 0
isnormal(Inf) = 0
isnormal(-Inf) = 0
isnormal(NaN) = 0
```

関連項目

[6]訳注：現行バージョンの Ch では、**isnormal** マクロは Windows プラットフォーム上でサポートされていません。

isunordered

構文

#include <**math.h**>
double isunordered(*real-floating x, real-floating y*);

目的

順序付けの有無をチェックします[7]。

戻り値

isunordered マクロは、引数が順序付けなしであるかどうかを判断します。

パラメータ

x 順序付けの有無をチェックする一方の値。

y 順序付けの有無をチェックする他方の値。

説明

isunordered マクロは、引数が順序付けられていない場合は 1 を返し、それ以外の場合は 0 を返します。

例

出力

関連項目

[7]訳注：現行バージョンの Ch では、**isunordered** マクロはいかなるプラットフォーム上でもサポートされていません。

ldexp

構文

#include <**math.h**>
double ldexp(**double** *x*, **int** *exp*);

目的

浮動小数点数値と 2 の累乗を乗算します。

戻り値

ldexp 関数は、$x \times 2^{\mathbf{exp}}$ の値を返します。

パラメータ

x $2^{\mathbf{exp}}$ 倍される値。

exp 2 の整数乗される指数。

説明

ldexp 関数は、浮動小数点数値と 2 の整数乗を乗算します。範囲エラーが発生することがあります。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 3.0;
    int i = 2;
    printf("ldexp(%f, %d) = %f\n", x, i, ldexp(x, i));
}
```

出力

```
ldexp(3.000000, 2) = 12.000000
```

関連項目

lgamma

構文

#include <**math.h**>
double lgamma(double *x***);**

目的

ガンマ関数の絶対値の自然対数値を計算します[8]。

戻り値

lgamma 関数はガンマ関数の絶対値の自然対数値を返します。

パラメータ

x ガンマ関数の値を求める値。

説明

lgamma 関数は、*x* のガンマ関数の絶対値の自然対数 $\log_e |\Gamma(x)|$ を計算します。*x* の絶対値が大きすぎる場合または小さすぎる場合、範囲エラーが発生することがあります。

例

```
/* The lgamma functions compute the natural  logarithm  of
       the  absolute value of Gamma of x. */
#include <stdio.h>
#include <math.h>

int main() {
    double x = -12.5;

    printf("ln(|Gamma (%2.1f)| is %2.1f\n",x,lgamma(x));
}
```

出力

```
ln(|Gamma (-12.5)| is -20.1
```

関連項目

[8]訳注：現行バージョンの Ch では、**lgamma** 関数は Windows プラットフォーム上でサポートされていません。

log

構文

#include <**math.h**>
double log(double *x*);

目的

e を底とする対数を計算します。

戻り値

log 関数は、*e* を底とする対数値を返します。

パラメータ

x 真数の値。

説明

log 関数は、*e* を底とする *x* の対数（自然対数）を計算します。引数が負の場合、領域エラーが発生します。引数がゼロの場合、範囲エラーが発生することがあります。この関数は、特殊な数字に関連する以下の値を返します。
— **log**(±0) は $-\infty$ を返します。
— **log**(1) は+0 を返します。
— **log**(*x*) は $x < 0$ の場合に NaN を返します。
— **log**($+\infty$) は $+\infty$ を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("log(x) = %f\n", log(x));
}
```

出力

```
log(x) = -1.203973
```

関連項目

log10

構文

#include <**math.h**>
double log10(**double** *x*);

目的

10 を底とする対数を計算します。

戻り値

log10 関数は、10 を底とする対数値を返します。

パラメータ

x 真数の値。

説明

log10 関数は、10 を底とする *x* の対数（常用対数）を計算します。引数が負の場合、領域エラーが発生します。引数がゼロの場合、範囲エラーが発生することがあります。この関数は、特殊な数字に関連する以下の値を返します。
— **log10**(± 0) は $-\infty$ を返します。
— **log10**(1) は+0 を返します。
— **log10**(*x*) は $x < 0$ の場合 NaN を返します。
— **log10**($+\infty$) は $+\infty$ を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("log10(x) = %f\n", log10(x));
}
```

出力

```
log10(x) = -0.522879
```

関連項目

log1p

構文

#include <**math.h**>
double log1p(**double** *x*);

目的

e を底とする対数を計算します。

戻り値

log1p 関数は、*e* を底とする引数+1 の対数値を返します。

パラメータ

x 真数-1 の値引数。-1 以上でなければなりません。。

説明

log1p 関数は、*e* を底とする引数+1 の対数（自然対数）を計算します。引数が-1 より小さい場合、領域エラーが発生します。引数が-1 の場合、範囲エラーが発生することがあります。この関数は、特殊な数字に関連する以下の値を返します。
— **log1p** (±0) は ±0 を返します。
— **log1p** (-1) は $-\infty$ を返します。
— **log1p** (*x*) は $x < -1$ の場合 NaN を返します。
— **log1p** ($+\infty$) は $+\infty$ を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("log(x) = %f\n", log(x));
}
```

出力

```
log(x) = -1.203973
```

関連項目

log2

構文

#include <**math.h**>
double log2(**double** *x*);

目的

2 を底とする対数を計算します。

戻り値

log2 関数は、2 を底とする対数値を返します。

パラメータ

x 真数の値。

説明

log2 関数は、2 を底とする *x* の対数を計算します。引数がゼロより小さい場合、領域エラーが発生します。引数がゼロの場合、範囲エラーが発生することがあります。この関数は、特殊な数字に関連する以下の値を返します。
— **log2** (±0) は $-\infty$ を返します。
— **log2** (*x*) は $x < 0$ の場合に NaN を返します。
— **log2** ($+\infty$) は $+\infty$ を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 2;
    printf("log2(x) = %f\n", log2(x));
}
```

出力

```
log2(x) = 1.000000
```

関連項目

logb

構文

#include <**math.h**>
double logb(**double** *x*);

目的

符号付き指数を返します。

戻り値

logb 関数は、*x* の符号付き指数を返します。

パラメータ

x 指数を求める値。

説明

logb 関数は、*x* の指数部を浮動小数点形式の符号付き整数値として抽出します。*x* が正規化されていない場合、正規化されているものとして扱います。したがって、正の無限大の *x* は次のようになります。

$$1 \leq x \times \mathbf{FLT\_RADIX}^{-\log(x)} \leq \mathbf{FLT\_RADIX}.$$

引数がゼロの場合、領域エラーが発生することがあります。この関数は、特殊な数字に関連する以下の値を返します。
— **logb** (±0) は $-\infty$ を返します。
— **logb** (±∞) は $+\infty$ を返します。

例

```
/*  The logb functions return the signed exponent of x. */
#include <stdlib.h>
#include <math.h>

int main() {
    double x = 10.0;
    printf("The result is %f\n",logb(x));
}
```

出力

```
The result is 3.000000
```

関連項目

lrint

構文
#include <**math.h**>
long int lrint(double *x*);
long long int llrint(double *x*);

目的
整数に丸めます[9]。

戻り値
lrint 関数および **llrint** 関数は、整数に丸めた値を返します。

パラメータ
x 整数に丸める値。

説明
lrint 関数および **llrint** 関数は、現在の丸め方向に従って、引数を最も近い整数値に丸めます。丸めた値が戻り値の型の範囲外である場合、数値結果は未指定になります。*x* の絶対値が大きすぎる場合、範囲エラーが発生することがあります。

例
出力
関連項目

[9]訳注：現行バージョンの Ch では、**lrint** 関数および **llrint** 関数はいかなるプラットフォーム上でもサポートされていません。

lround

構文
#include <**math.h**>
long int lround(double *x*);
long long int llround(double *x*);

目的
整数に丸めます[10]。

戻り値
lround 関数および **llround** 関数は、整数に丸めた値を返します。

パラメータ
x 整数に丸める値。

説明
lround 関数および **llround** 関数は、引数を最も近い整数値に丸めます。値が中央にある場合は、現在の丸め方向に関係なく、ゼロから遠い方に丸めます。丸めた値が戻り値の型の範囲外である場合、数値結果は未指定になります。*x* の絶対値が大きすぎる場合、範囲エラーが発生することがあります。

例
出力
関連項目

[10]訳注：現行バージョンの Ch では、**lround** 関数および **llround** 関数はいかなるプラットフォーム上でもサポートされていません。

modf

構文

#include <**math.h**>
double modf(double *value*, **double** **iptr*);

目的

小数値を返します。

戻り値

modf 関数は、*value* の符号付き小数部の値を返します。

パラメータ

value 小数と整数との分解される値。

iptr 整数部を格納する実数型変数へのポインタ引数。実数型変数の具体的な型は *value* の型と同じでなければなりません。

説明

modf 関数は、引数 *value* を、それぞれ引数と同じ型と符号を持つ整数部と小数部に分けます。この関数は、*iptr* が指すオブジェクトに浮動小数点形式で整数部を格納します。この関数は、特殊な数字に関連する以下の値を返します。
— **modf**(*value*, *iptr*) は、引数 *value* と同じ符号を付けて結果を返します。
— **modf**(±∞, *iptr*) は ±0 を返し、*iptr* が指すオブジェクトに ±∞ を格納します。
— **modf** の引数が NaN の場合、*iptr* が指すオブジェクトに NaN を格納して、NaN を返します。

例

出力

関連項目

nan

構文

#include <**math.h**>
double nan(**const char** **tagp*);

目的

NaN を返します[11]。

戻り値

nan 関数は、*tagp* で指定された表現でシグナルなしの NaN を返します。シグナルなしの NaN をサポートしていない実装ではゼロを返します。

パラメータ

tagp 引数。

説明

nan(”*n-char-sequence***”)** の呼び出しは **strtod(**`"nan(n-char-sequence)", (char**) NULL`**)** と同じです。また、**nan(””)** の呼び出しは、**strtod**`"nan(), (char**) NULL`）と同じです。tagp が n-char-sequence 文字列や空の文字列を指していない場合、この呼び出しは、**strtod**`("nan", (char**) NULL`）と同じになります。**nanf** および **nanl** の呼び出しは、対応する **strtof** および **strtold** の呼び出しと同じです。

例

出力

関連項目

[11]訳注：現行バージョンの Ch では、**nan** 関数はいかなるプラットフォーム上でもサポートされていません。

nearbyint

構文

#include <**math.h**>
double nearbyint(double *x*);

目的

整数に丸めます[12]。

戻り値

nearbyint 関数は、整数に丸めた値を返します。

パラメータ

x 整数に丸められる値。

説明

nearbyint 関数は、現在の丸め方向に従って引数を浮動小数点形式の整数値に丸めます。この関数では *inexact* 例外は発生しません。

例

出力

関連項目

[12]訳注：現行バージョンの Ch では、**nearbyint** 関数はいかなるプラットフォーム上でもサポートされていません。

nextafter

構文

#include <**math.h**>
double nextafter(double *x*, **double** *y*);

目的

次の値を返します。

戻り値

nextafter 関数は、*y* の方向で *x* の次に表現可能な値を指定された形式で返します。

パラメータ

x 開始点の値。

y 開始点からみての向きを決めるための任意の値。

説明

nextafter 関数は、この関数型の *y* の方向で *x* の次に表現可能な値を判断します。このとき、*x* と *y* は最初に関数の型に変換されます。*x* が *y* に等しい場合、 **nextafter** 関数は *y* を返します。*x* の絶対値がこの型で表現可能な最大の有限値である場合、または、結果が無限大かこの型で表現できない場合、範囲エラーが発生することがあります。

例

```
/* a sample program that computes the next representable value */
#include <math.h>
#include <stdio.h>

int main() {
    double x, y, z;

    x = 5.0;
    y = 10.0;
    z = nextafter(x, y);
    printf("%f is the next representable value.\n",z);
}
```

出力

```
5.000000 is the next representable value.
```

関連項目

nexttoward

構文
#include <**math.h**>
double nexttoward(**double** *x*, **long double** *y*);

目的
小数値を返します[13]。

戻り値

パラメータ
x 開始点の値。

y 開始点からみての向きを決めるための任意の値。

説明
nexttoward 関数は、2 番目のパラメータが **long double** 型であること以外は、 **nextafter** 関数と同じです。

例
出力
関連項目

[13]訳注：現行バージョンの Ch では、**nexttoward** 関数はいかなるプラットフォーム上でもサポートされていません。

pow

構文

#include <**math.h**>
type pow(**type** *x*, **type** *y*);
double pow(**double** *x*, **double** *y*);

目的

累乗の値を計算します。

戻り値

pow 関数は、*x* を *y* 乗した値を返します。

パラメータ

x 底 (基数) の値。

y 冪指数の値。

説明

pow 関数は、*x* の *y* 乗を計算します。*x* が負で、*y* が有限の非整数値の場合、領域エラーが発生します。*x* がゼロで *y* がゼロ以下のとき、結果を表現できない場合に領域エラーが発生します。この関数は、特殊な数字に関連する以下の値を返します。
— **pow**$(x, \pm 0)$ は、NaN を含むすべての x に対して 1 を返します。
— **pow**$(x, +\infty)$ は、$|x| > 1$ の場合に $+\infty$ を返します。
— **pow**$(x, +\infty)$ は、$|x| < 1$ の場合に+0 を返します。
— **pow**$(x, -\infty)$ は、$|x| > 1$ の場合に+0 を返します。
— **pow**$(x, -\infty)$ は、$|x| < 1$ の場合に $+\infty$ を返します。
— **pow**$(+\infty, y)$ は、$|y| > 0$ の場合に $+\infty$ を返します。
— **pow**$(+\infty, y)$ は、$|y| < 0$ の場合に+0 を返します。
— **pow**$(-\infty, y)$ は、y が 0 より大きい奇数の整数である場合に $-\infty$ を返します。
— **pow**$(-\infty, y)$ は、y が 0 より大きく、奇数の整数でない場合に $+\infty$ を返します。
— **pow**$(-\infty, y)$ は、y が 0 より小さく奇数の整数である場合に-0 を返します。
— **pow**$(-\infty, y)$ は、y より小さく、奇数の整数でない場合に+0 を返します。
— **pow**$(\pm 1, \pm\infty)$ は NaN を返します。
— **pow**(x, y) は、x が 0 より小さい有限値である場合、および¿y が整数でない有限値である場合に $+\infty$ を返します。
— **pow**$(\pm 0, y)$ は、y が 0 より小さく、奇数に整数である場合に $\pm\infty$ を返します。
— **pow**$(\pm 0, y)$ は、y が 0 より小さく、奇数の整数でない場合に $+\infty$ を返します。
— **pow**$(\pm 0, y)$ は、y が 0 より大きい奇数の整数である場合に $\pm$ 0 を返します。
— **pow**$(\pm 0, y)$ は、y が 0 より大きく、奇数の整数でない場合に+0 を返します。

例

```
#include <stdio.h>
#include <math.h>

int main() {
    double x = 2.0;
    double y = 3.0;
    printf("pow(%f %f) = %f\n", x, y, pow(x, y));
}
```

出力

```
pow(2.000000 3.000000) = 8.000000
```

関連項目

remainder

構文

#include <**math.h**>
double remainder(**double** *x*, **double** *y*);

目的

剰余を計算します[14]。

戻り値

remainder 関数は、*x* REM *y* の値を返します。

パラメータ

x 被除数の値。

y 除数の値。

説明

remainder 関数は、剰余 (*x* REM *y*) を計算します。

例

```
/* The function returns the value of r = x REM y.  r=x-ny, where n is the
   integer nearest the exact value  of  x/y. */
#include <math.h>
#include <stdio.h>

int main() {
    double x, y, z;

    x = 5.0;
    y = 2.0;
    z =remainder(x, y);
    printf("x =  %f\n", x);
    printf("y = %f\n", y);
    printf("Remainder = %f.\n",z);
}
```

出力

```
x =  5.000000
y = 2.000000
Remainder = 1.000000.
```

関連項目

[14]訳注：現行バージョンの Ch では、remainder 関数は Windows プラットフォーム上でサポートされていません。

remquo

構文
#include <**math.h**>
double remquo(**double** *x*, **double** *y*, **int*****quo*);

目的
小数値を返します[15]。

戻り値
remquo 関数は、*x* REM *y* の値を返します。

パラメータ
x 被除数の値

y 除数の値。

quo 剰余を格納する整数型変数へのポインタ。

説明
remquo 関数は、 **remainder** 関数と同様に剰余を計算します。この関数は、*quo* が指すオブジェクトに、*x*/*y* の符号を持ち、*x*/*y* の整数商の絶対値に対して 2^n を法とする合同な値を格納します。*n* は実装で定義される 3 以上の整数です。

例
出力
関連項目

[15]訳注：現行バージョンの Ch では、**remquo** 関数はいかなるプラットフォーム上でもサポートされていません。

rint

構文

#include <**math.h**>
double rint(double *x*);

目的

整数に丸めます[16]。

戻り値

rint 関数は、整数に丸めた値を返します。

パラメータ

x 整数に丸められる値。

説明

rint 関数は、次の点のみが **nearbyint** 関数と異なります。つまり、 **rint** 関数では結果の値が引数と異なる場合に inexact 例外が発生することがあります。この関数は、特殊な数字に関連する以下の値を返します。
— **rint**(±0) は、すべての丸め方向に対して ±0 を返します。
— **rint**(±∞) は、すべての丸め方向に対して ±∞ を返します。

例

```
/* a sample program that rounds a double number. */
#include <stdlib.h>
#include <math.h>

int main() {
    double x = 213.645;
    printf("%3.3f is rounded to %f \n",x,rint(x));
}
```

出力

```
213.645 is rounded to 214.000000
```

関連項目

[16]訳注：現行バージョンの Ch では、**rint** 関数は Windows プラットフォーム上でサポートされていません。

round

構文
#include <**math.h**>
double round(double *x***);**

目的
整数に丸めます[17]。

戻り値
round 関数は、整数に丸めた値を返します。

パラメータ
x 整数に丸められる値。

説明
round 関数は、引数に最も近い整数値に丸めます。値が中央にある場合は、現在の丸め方向に関係なく、ゼロから遠い方に丸めます。

例
出力
関連項目

[17]訳注：現行バージョンの Ch では、**round** 関数はいかなるプラットフォーム上でもサポートされていません。

scalbn および scalbln

構文

#include <**math.h**>
double scalbn(**double** *x*, **int** *n*);
double scalbln(**double** *x*, **int** *n*);

目的

$x\times$ $\mathbf{FLT\_RADIX}^n$ の値を返します[18]。

戻り値

scalbn 関数および **scalbln** 関数は $x\times$ $\mathbf{FLT\_RADIX}^n$ の値を返します。

パラメータ

x $\mathbf{FLT\_RADIX}^n$ に乗じる値。

n **FLT_RADIX** の冪指数。

説明

scalbn 関数および **scalbln** 関数は、$\mathbf{FLT\_RADIX}^n$ を明示的に計算することにより、通常とは異なる効率的な方法で $x\times$ $\mathbf{FLT\_RADIX}^n$ を計算します。この関数は、特殊な数字に関連する以下の値を返します。
— **scalbn**(x, n) は、x が無限大またはゼロの場合に x を返します。
— **scalbn**$(x, 0)$ は x を返します。

例

```
/* a sample program that decides which of the two values is a max. */
#include <math.h>
#include <stdio.h>

int main() {
    double x, z;
    int y;
    x = 5.0;
    y = 2;
    z =scalbn(x, y);
    printf("Value 1: %f\n", x);
    printf("Value 2: %d\n", y);
    printf("%f is the maximum numeric value.\n",z);
}
```

出力

```
Value 1: 5.000000
Value 2: 2
20.000000 is the maximum numeric value.
```

[18]訳注：現行バージョンの Ch では、**scalbn** 関数および **scalbln** 関数は Windows プラットフォーム上でサポートされていません。

関連項目

signbit

構文

#include <**math.h**>
int signbit(double *x***);**

目的

引数が負の値であるかどうかを判断します[19]。

戻り値

signbit マクロは、引数が負の値の場合のみ、ゼロ以外の値を返します。

パラメータ

x 正負の符号をテストされる値。

説明

signbit マクロは、引数の値が負であるかどうかを判断します。

例

```
#include <stdio.h>
#include <math.h>

int main() {
    printf("signbit(3) = %d\n",  signbit(3));
    printf("signbit(-3) = %d\n",  signbit(-3));
}
```

出力

関連項目

[19]訳注：現行バージョンの Ch では、**signbit** マクロは Windows プラットフォーム上でサポートされていません。

sin

構文

#include <**math.h**>
type sin(**type** *x*);
double sin(**double** *x*);

目的

サインを計算します。

戻り値

sin 関数は、サインの値を返します。

パラメータ

x サインを計算する値。

説明

sin 関数は、ラジアンで測定された *x* のサインを計算します。この関数は、特殊な数字に関連する以下の値を返します。
The function returns the following values related to special numbers:
— **sin**(±0) は ±0 を返します。
— **sin**(±∞) は NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("sin(x) = %f\n", sin(x));
}
```

出力

```
sin(x) = 0.295520
```

関連項目

asin(), **asinh**(), **sinh**().

sinh

構文

#include <**math.h**>
type sinh(type *x*);
double sinh(double *x*);

目的

双曲線サインを計算します。

戻り値

sinh 関数は、双曲線サインの値を返します。

パラメータ

x 双曲線サインを計算する値。

説明

sinh 関数は、*x* の双曲線サインを計算します。*x* の絶対値が大きすぎる場合、範囲エラーが発生します。この関数は、特殊な数字に関連する以下の値を返します。
— **sinh(**±0**)** は ±0 を返します。
— **sinh(**±∞**)** は ±∞ を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("sinh(x) = %f\n", sinh(x));
}
```

出力

```
sinh(x) = 0.304520
```

関連項目

sin(), **asinh**(), **asin**().

sqrt

構文

#include <**math.h**>
double sqrt(**double** *x*);

目的

平方根を計算します。

戻り値

sqrt 関数は平方根の値を返します。

パラメータ

x 平方根を計算する値。ゼロまたは正の値でなければなりません。

説明

sqrt 関数は、*x* の負でない平方根を計算します。引数がゼロより小さい場合、領域エラーが発生します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("sqrt(x) = %f\n", sqrt(x));
}
```

出力

```
sqrt(x) = 0.547723
```

関連項目

tan

構文

#include <**math.h**>
type tan(**type** *x*);
double tan(**double** *x*);

目的

タンジェントを計算します。

戻り値

tan 関数はタンジェントの値を返します。

パラメータ

x タンジェントを計算する値。

説明

tan 関数は、ラジアンで測定された *x* のタンジェントを計算します。この関数は、特殊な数字に関連する以下の値を返します。
— **tan**(±0) は ±0 を返します。
— **tan**(±∞) は NaN を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("tan(x) = %f\n", tan(x));
}
```

出力

```
tan(x) = 0.309336
```

関連項目

atan(), **atanh**(), **tanh**().

tanh

構文

#include <**math.h**>
type tanh(**type** *x*);
double tanh(**double** *x*);

目的

双曲線タンジェントを計算します。

戻り値

tanh 関数は双曲線タンジェントの値を返します。

パラメータ

x 双曲線タンジェントを計算する値。

説明

tanh 関数は *x* の双曲線タンジェントを計算します。この関数は、特殊な数字に関連する以下の値を返します。

— **tanh**(±0) は ±0 を返します。
— **tanh**(±∞) は ±1 を返します。

例

```
#include <math.h>
#include <stdio.h>

int main () {
    double x = 0.3;
    printf("tanh(x) = %f\n", tanh(x));
}
```

出力

```
tanh(x) = 0.291313
```

関連項目

tan(), **atanh**(), **atan**().

tgamma

構文
#include <**math.h**>
double tgamma(**double** *x*);

目的
ガンマ値を計算します[20]。

Return Value
tgamma 関数はガンマ関数値を返します。

パラメータ
x ガンマ関数を計算する値。

説明
tgamma 関数は、*x* のガンマ関数 $\Gamma(x)$ を計算します。*x* が負の整数またはゼロの場合、領域エラーが発生します。*x* の絶対値が大きすぎる場合または小さすぎる場合、範囲エラーが発生することがあります。この関数は、特殊な数字に関連する以下の値を返します。
— **tgamma**($+\infty$) は $+\infty$ を返します。
— **tgamma**(x) は、x が負の整数またはゼロの場合に $+\infty$ を返します。
— **tgamma**($-\infty$) は NaN を返します。

例
出力
関連項目

[20]訳注：現行バージョンの Ch では、**tgamma** 関数はいかなるプラットフォーム上でもサポートされていません。

trunc

構文
#include <**math.h**>
double trunc(**double** *x*);

目的
整数に切り捨てます[21]。

戻り値
trunc 関数は整数値を切り捨てます。

パラメータ
x 整数に切り捨てられる値。

説明
trunc 関数は、引数の絶対値以下の最も近い整数値 (浮動小数点形式) に引数を丸めます。

例
出力
関連項目

[21]訳注：現行バージョンの Ch では、**trunc** 関数はいかなるプラットフォーム上でもサポートされていません。

第11章　数値解析 — <numeric.h>

ヘッダーファイル **numeric.h** は、Ch Professional Edition でのみ利用可能な高度な数値解析関数の定義を含んでいます。この中には、ヘッダーファイル **math.h**、**stdarg.h**、**array.h** および **dlfcn.h** が含まれています。数値解析用に設計されている高度な関数は、以下のとおりです。

関数	説明
balance()	一般的な実数型の行列を調整し、固有値計算の精度を高めます。
ccompanionmatrix()	複素数型の随伴行列を計算します。
cdeterminant()	複素行列の行列式を求めます。
cdiagonal()	複素行列の対角ベクトルを求めます。
cdiagonalmatrix()	複素ベクトルから対角行列を作成します。
cfevalarray()	複素配列関数を評価します。
cfunm()	一般的な複素数型の行列関数を評価します。
charpolycoef()	行列の固有多項式の係数を計算します。
choldecomp()	対称正定値行列 A の Cholesky 分解を計算します。
cinverse()	複素数型の正方行列の逆行列を求めます。
clinsolve()	LU 分解で複素連立一次方程式の解を求めます。
cmean()	複素配列のすべての要素の平均値と各行の平均値を計算します。
combination()	*n* 個から一度に *k* 個を選ぶ組み合わせの数を計算します。
companionmatrix()	随伴行列を計算します。
complexsolve()	複素方程式の解を求めます。
complexsolvePP()	複素方程式の解を求めます。
complexsolvePR()	複素方程式の解を求めます。
complexsolveRP()	複素方程式の解を求めます。
complexsolveRR()	複素方程式の解を求めます。
complexsolveRRz()	複素方程式の解を求めます。
condnum()	行列の条件数を計算します。
conv()	1 次元離散 Fourier 変換 (DFT) で畳み込みを計算します。
conv2()	2 次元離散 Fourier 変換 (DFT) で畳み込みを計算します。
corrcoef()	相関係数を計算します。
correlation2()	2 次元相関係数を求めます。
covariance()	共分散行列を求めます。
cpolyeval()	複素多項式の値を複素数のデータ点で計算します。

cproduct()	すべての要素の積または任意の次元を持つ複素配列の要素の積、および 2 次元複素配列の各行の要素の積を計算します。
covariance()	共分散行列を求めます。
cross()	ベクトルの外積を計算します。
csum()	すべての要素の和または任意の次元の複素配列の要素の和、および 2 次元複素配列の各行の要素の和を計算します。
ctrace()	複素行列の対角要素の合計を計算します。
ctriangularmatrix()	複素行列の上三角部または下三角部を求めます。
cumprod()	配列内の要素の累積積を計算します。
cumsum()	配列内の要素の累積和を計算します。
curvefit()	データ点 x と y のセットを、指定したベース関数の線形結合へ一致させます。
deconv()	1 次元離散型 Fourier 変換 (DFT) で逆重畳を計算します。
derivative()	所定のデータ点での関数の導関数を数値計算します。
derivatives()	複数のデータ点での関数の導関数を数値計算します。
determinant()	行列の行列式を求めます。
diagonal()	行列の対角ベクトルを求めます。
diagonalmatrix()	ベクトルの対角行列を作成します。
difference()	配列の隣接要素の差を計算します。
dot()	ベクトルの内積を計算します。
eigen()	固有値と固有ベクトルを求めます。
eigensystem()	固有値と固有ベクトルを求めます。
expm()	行列の指数を計算します。
factorial()	階乗を計算します。
fevalarray()	配列関数を評価します。
fft()	N 次元の高速 Fourier 変換 (FFT) を計算します。
filter()	データをフィルタ処理します。
filter2()	2 次元離散型 Fourier 変換で FIR フィルタ処理を行います。
findvalue()	配列の要素のゼロ以外のインデックスを求めます。
fliplr()	行列を左右に反転します。
flipud()	行列を上下に反転します。
fminimum()	1 次元関数の最小値を求め、最小値に対応する位置を特定します。
fminimums()	n 次元関数の最小の位置と値を求めます。
fsolve()	非線形連立方程式のゼロ位置を求めます。
funm()	一般的な実数型の行列関数を評価します。
fzero()	1 つの変数を含む非線形関数のゼロ点を求めます。
gcd()	整数型の 2 つの配列の対応する値の最大公約数を得ます。
getnum()	コンソールから既定値の数を取得します。
hessdecomp()	直交行列またはユニタリ行列の相似変換で、一般的な実数型の行列を上位の Hessenberg 形式に還元します。
histogram()	ヒストグラムを計算し、プロット作成します。
householdermatrix()	Householder 行列を求めます。

identitymatrix()	単位行列を作成します。
ifft()	N 次元の逆高速 Fourier 変換 (FFT) を計算します。
integral1()	1 次元関数の数値積分を行います。
integral2()	2 次元関数の数値積分を行います。積分範囲は 2 つの変数共に固定です。
integral3()	3 次元関数の数値積分を行います。積分範囲は 3 つの変数全てについて固定です。
integration2()	2 次元関数の数値積分を行います。一方の変数に対する積分範囲は他方の変数に依存します。
integration3()	3 次元関数の数値積分を行います。2 つの変数に対する積分範囲は残り 1 つの変数に依存します。
interp1()	1 次元補間を行います。
interp2()	2 次元補間を行います。
inverse()	正方行列の逆行列を求めます。
lcm()	整数型の 2 つの配列の対応する値の最小公倍数を求めます。
lindata()	等間隔の連続データを生成します。
linsolve()	LU 分解で連立一次方程式の解を求めます。
linspace()	等間隔の連続配列を生成します。非推奨。代わりに **lindata**() を使用します。
llsqcovsolve()	線形最小 2 乗法で、既知の共分散を含む連立一次方程式の解を求めます。
llsqnonnegsolve()	線形最小 2 乗法で、ゼロ以外の値を含む連立一次方程式の解を求めます。
llsqsolve()	最小 2 乗法で一次方程式の解を求めます。
logm()	行列の自然対数を計算します。
logdata()	対数軸上で等間隔に区切られたデータを生成します。
logspace()	対数軸上で等間隔に区切られた配列を生成します。非推奨。代わりに **logdata**() を使用します。
ludecomp()	一般的な n x m 行列を LU 分解します。
maxloc()	配列の最大値の位置を求めます。
maxv()	行列の各行の要素の最大値を求めます。
mean()	配列のすべての要素の平均値と各行の平均値を計算します。
median()	配列のすべての要素の中央値と各要素の中央値を計算します。
minloc()	配列の最小値の位置を計算します。
minv()	2 次元配列の各行の最小値を計算します。
norm()	ベクトルと行列のノルムを計算します。
nullspace()	行列の null 空間を計算します。
oderk()	ルンゲクッタ法を使用して、常微分方程式の解を求めます。
oderungekutta()	ルンゲクッタ法を使用して、常微分方程式の解を求めます。非推奨。
odesolve()	常微分方程式の解を求めます。非推奨。
orthonormalbase()	行列の直交ベースを計算します。
pinverse()	行列の Moore-Penrose 擬似逆行列を計算します。
polyder()	多項式の導関数を求めます。
polyder2()	2 つの多項式の積または商の導関数を計算します。
polyeval()	多項式の値とその導関数を計算します。
polyevalarray()	点列で多項式を評価します。

polyevalm() 行列多項式の値を計算します。
polyfit() データ点のセットを多項式関数に一致させます。
product() 任意の次元を持つ配列のすべての要素の積、または 2 次元配列の各列の要素の積を計算
qrdecomp() 行列の直交三角形の QR 分解を計算します。
qrdelete() QR 分解された行列から列を削除します。
qrinsert() QR 分解された行列に列を挿入します。
rank() 行列のランクを計算します。
residue() 部分端数の展開または剰余を計算します。
rcondnum() 行列の逆比例条件を見積ります。
roots() 多項式の根を計算します。
rot90() 行列を 90 度回転させます。
rsf2csf() 実数 Schur 形式を複素数 Schur 形式に変改します。
specialmatrix() 特別な行列を生成します。
- **Cauchy** Cauchy 行列を生成します。
- **ChebyshevVandermonde** Chebyshev 多項式のための Vandermonde に似た行列を生成します。
- **Chow** Chow 行列を生成します。
- **Circul** Circul 行列を生成します。
- **Clement** Clement 行列を生成します。
- **DenavitHartenberg** Denavit-Hartenberg 行列を生成します。
- **DenavitHartenberg**2 Denavit-Hartenberg 行列を生成します。
- **Dramadah** Dramadah 行列を生成します。
- **Fiedler** Fiedler 行列を生成します。
- **Frank** Frank 行列を生成します。
- **Gear** Gear 行列を生成します。
- **Hadamard** Hadamard 行列を生成します。
- **Hankel** Hankel 行列を生成します。
- **Hilbert** Hilbert 行列を生成します。
- **InverseHilbert** 逆 Hilbert 行列を生成します。
- **Magic** Magic 行列を生成します。
- **Pascal** Pascal 行列を生成します。
- **Rosser** Rosser 行列を生成します。
- **Toeplitz** Toeplitz 行列を生成します。
- **Vandermonde** Vandermonde 行列を生成します。
- **Wilkinson** Wilkinson 行列を生成します。

schurdecomp() Schur 分解を計算します。
sign() 引数の符号を求めます
sort() 要素を昇順に並べ換えて、ランク付けします。
sqrtm() マトリクスの平方根を計算します。
std() データセットの標準偏差を計算します。
sum() 配列のすべての要素の合計または各行の要素の合計を計算します。
svd() 特異値を分解します。

trace()	対角要素の合計を計算します。
triangularmatrix()	行列の上三角または下三角を求めます。
unwrap()	π より大きい絶対ジャンプを $2 * \pi$ 補数へ変えることにより、 入力される配列 x の各要素のラジアン位相をアンラップします。
urand()	一様乱数を生成します。
xcorr()	1 次元相互相関ベクトルを求めます。

---

balance

Synopsis
#include <**numeric.h**>
int balance(**array double complex** *a*[&][&], **array double complex** *b*[&][&], ...
/* [**array double complex** *d*[&][&]] */);

Syntax
balance(*a*, *b*); balance(*a*, *b*, *d*);
Purpose
Balances a general matrix to improve accuracy of computed eigenvalues.

Return Value
This function returns 0 on success and negative value on failure.

Parameters
a An $n \times n$ square matrix to be balanced.

l An output square matrix which contains the balanced matrix of matrix *a*.

d An optional output square matrix which contains a diagonal matrix.

Description
This function computes the balance matrix *b* and diagonal matrix *d* from input matrix *a* so that $b = d^{-1} * a * d$. If *a* is symmetric, then $b == a$ and *d* is the identity matrix.
In this function, square matrix *a* could be any supported arithmetic data type. Output *b* is the same dimension and data type as input *a*. If the input *a* is a real matrix, the optional output *d* shall only be **double** data type. If the input *a* is a complex matrix, *d* shall be **complex** or **double complex**.

Example1
balance a real general matrix.

```
#include <numeric.h>
int main() {
     int m = 5;

     array double a[5][5] = {     1,  10, 100, 1000, 10000,
                                0.1,   1,  10,  100,  1000,
                               0.01, 0.1,   1,   10,   100,
                              0.001,0.01, 0.1,    1,    10,
                             0.0001,0.001,0.01, 0.1,     1};

     int status;
     array double b[m][m], d[m][m];
     status = balance(a, b, d);
     if (status == 0) {
        printf("a =\n%f\n", a);
        printf("b =\n%f\n", b);
```

```
        printf("d =\n%f\n", d);
        printf("inverse(d)*a*d - b =\n%f\n", inverse(d)*a*d-b);
     }
     else
        printf("error: calculation error in balance().\n");

}
```

Output1

```
a =
1.000000 10.000000 100.000000 1000.000000 10000.000000
0.100000 1.000000 10.000000 100.000000 1000.000000
0.010000 0.100000 1.000000 10.000000 100.000000
0.001000 0.010000 0.100000 1.000000 10.000000
0.000100 0.001000 0.010000 0.100000 1.000000

b =
1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000

d =
1000.000000 0.000000 0.000000 0.000000 0.000000
0.000000 100.000000 0.000000 0.000000 0.000000
0.000000 0.000000 10.000000 0.000000 0.000000
0.000000 0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.100000

inverse(d)*a*d - b =
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
-0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
-0.000000 0.000000 0.000000 0.000000 0.000000
```

Example2
Balance a complex general matrix.

```
#include <numeric.h>

int main() {
     int m = 4;

     array double complex a[4][4] = {     0, 0, complex(1,1), 0,
                                          0, 0, 0, complex(1,1),
                                    complex(11,1),complex(10,1), 0, 0,
                                    complex(10,1),complex(11,1), 0, 0};

     int status;
     array double complex b[m][m];
     array double d[m][m];
     status = balance(a, b, d);
     if (status == 0) {
        printf("a =\n%4.2f\n", a);
        printf("b =\n%4.2f\n", b);
```

```
        printf("d =\n%4.2f\n", d);
        printf("inverse(d)*a*d -b =\n%4.2f\n", inverse(d)*a*d-b);
     }
     else
        printf("error: calculation error in balance().\n");

}
```

Output2

```
a =
complex(0.00,0.00) complex(0.00,0.00) complex(1.00,1.00) complex(0.00,0.00)

complex(0.00,0.00) complex(0.00,0.00) complex(0.00,0.00) complex(1.00,1.00)

complex(11.00,1.00) complex(10.00,1.00) complex(0.00,0.00) complex(0.00,0.00)

complex(10.00,1.00) complex(11.00,1.00) complex(0.00,0.00) complex(0.00,0.00)


b =
complex(0.00,0.00) complex(0.00,0.00) complex(10.00,10.00) complex(0.00,0.00)

complex(0.00,0.00) complex(0.00,0.00) complex(0.00,0.00) complex(10.00,10.00)

complex(1.10,0.10) complex(1.00,0.10) complex(0.00,0.00) complex(0.00,0.00)

complex(1.00,0.10) complex(1.10,0.10) complex(0.00,0.00) complex(0.00,0.00)


d =
0.10 0.00 0.00 0.00
0.00 0.10 0.00 0.00
0.00 0.00 1.00 0.00
0.00 0.00 0.00 1.00

inverse(d)*a*d -b =
complex(0.00,0.00) complex(0.00,0.00) complex(0.00,0.00) complex(0.00,0.00)

complex(0.00,0.00) complex(0.00,0.00) complex(0.00,0.00) complex(0.00,0.00)

complex(0.00,0.00) complex(0.00,0.00) complex(0.00,0.00) complex(0.00,0.00)

complex(0.00,0.00) complex(0.00,0.00) complex(0.00,0.00) complex(0.00,0.00)
```

See Also

eigensystem().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

ccompanionmatrix

Synopsis
#include <**numeric.h**>
array double complex ccompanionmatrix(array double complex *v*[&])[:][:];

Purpose
Find companion matrix with complex number.

Return Value
This function returns the complex companion matrix.

Parameters
v Input array containing the complex coefficients of a polynomial.

Description
This function returns the corresponding complex companion matrix of the complex array *v* which contains the coefficients of a polynomial. The eigenvalues of companion matrix are roots of the polynomial.

Example

```
#include <numeric.h>
int main() {
    array double complex v[4] = {complex(1,2),3,4,5}; /* (1+i2)x^3 + 3x^2 + 4x + 5  */
    int n = 4;
    array double complex a[n-1][n-1];

    a = ccompanionmatrix(v);
    printf("ccompanionmatrix(a) =\n%f\n", a);
}
```

Output

```
ccompanionmatrix(a) =
complex(-0.600000,1.200000) complex(-0.800000,1.600000) complex(-1.000000,2.000000)

complex(1.000000,0.000000) complex(0.000000,0.000000) complex(0.000000,0.000000)

complex(0.000000,0.000000) complex(1.000000,0.000000) complex(0.000000,0.000000)
```

See Also
companionmatrix(), **roots**(), **eigensystem**(), **polycoef**().

References

cdeterminant

Synopsis
#include <**numeric.h**>
double complex cdeterminant(**array double complex** *a*[&][&]);

Purpose
Calculate the determinant of a complex matrix.

Return Value
This function returns the determinant of the complex matrix *a*.

Parameters
a The input complex matrix.

Description
The function **cdeterminant**() returns the determinant of complex matrix *a*. If the input matrix is not a square matrix, the function will return NaN.

Example
Calculate the determinant of complex matrices with different data type.

```
#include <numeric.h>
int main() {
    array double complex a[2][2] = {complex(1,2), 4,
                                    3, 7};
    /* a2 is an ill-detition matrix  */
    array double a2[2][2] = {2, 4,
                             2.001, 4.0001};
    /* a3 is singular */
    array double a3[2][2] = {2, 4,
                             4, 8};
    array complex b[2][2] = {2, 4,
                             3, 7};
    array double c[3][3] =  {-1,5,6,
                             3,-6,1,
                             6,8, 9} ; /* n-by-n matrix */
    double complex det;

    det = cdeterminant(a);
    printf("cdeterminant(a) = %g\n", det);
    det = cdeterminant(a2);
    printf("cdeterminant(a2) = %g\n", det);
    det = cdeterminant(a3);
    printf("cdeterminant(a3) = %g\n", det);
    det = cdeterminant(b);
    printf("cdeterminant(b) = %g\n", det);
    det = cdeterminant(c);
    printf("cdeterminant(c) = %g\n", det);
}
```

Output

```
cdeterminant(a) = complex(-5,14)
cdeterminant(a2) = complex(-0.0038,-0)
cdeterminant(a3) = complex(-0,-0)
cdeterminant(b) = complex(-2,0)
cdeterminant(c) = complex(317,-0)
```

See Also

determinant(), **inverse**(), **diagonal**(), **ludecomp**(), **rcondnum**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

cdiagonal

Synopsis
#include <**numeric.h**>
array double complex cdiagonal(**array double complex** *a*[&][&],... /* [**int** *k*] */)[:];

Syntax
cdiagonal(*a*);
cdiagonal(*a*, *k*);

Purpose
Form a vector with diagonals of a complex matrix.

Return Value
This function returns a column vector formed from the elements of the *k*th diagonal of the complex matrix *a*.

Parameters
a An input complex matrix.

k An input integer indicating which element the vector is formed from.

Description
The function **cdiagonal**() produces a row vector formed from the elements of *k*th diagonal of the complex matrix *a*.

Example1

```
#include <numeric.h>
int main() {
    array double complex a[3][3] = {complex(1,2), 2, 3,
                                    4, 5, 6,
                                    7, 8, 9};
    int n =3, k = 1;
    array double complex d[n], d2[n-(abs(k))];

    d = cdiagonal(a);
    printf("cdiagonal(a) = %0.2f\n", d);

    d2 = cdiagonal(a, k);
    printf("cdiagonal(a, 1) = %0.2f\n", d2);

    k = -1;
    d2 = cdiagonal(a, k);
    printf("cdiagonal(a, -1) = %0.2f\n", d2);
}
```

Output1

```
cdiagonal(a) = complex(1.00,2.00) complex(5.00,0.00) complex(9.00,0.00)

cdiagonal(a, 1) = complex(2.00,0.00) complex(6.00,0.00)

cdiagonal(a, -1) = complex(4.00,0.00) complex(8.00,0.00)
```

Example2

```
#include <numeric.h>
int main() {
    array double complex a[4][3] =  {complex(1,2), 2, 3,
                              4, 5, 6,
                              7, 8, 9,
                              4, 4, 4};
    int m=4, n = 3, k1 = 1, k2 = -1;
    array double complex d[min(m, n)];
    array double complex d1[min(m, n-k1)];
    array double complex d2[min(m+k2, n)];

    d = cdiagonal(a);
    printf("cdiagonal(a) = %0.2f\n", d);

    d1 = cdiagonal(a, k1);
    printf("cdiagonal(a, 1) = %0.2f\n", d1);

    d2 = cdiagonal(a, k2);
    printf("cdiagonal(a, -1) = %0.2f\n", d2);
}
```

Output2

```
cdiagonal(a) = complex(1.00,2.00) complex(5.00,0.00) complex(9.00,0.00)

cdiagonal(a, 1) = complex(2.00,0.00) complex(6.00,0.00)

cdiagonal(a, -1) = complex(4.00,0.00) complex(8.00,0.00) complex(4.00,0.00)
```

Example3

```
#include <numeric.h>
int main() {
    array double complex a[3][4] =  {complex(1,2), 2, 3, 4,
                              5, 6, 7, 8,
                              9, 4, 4, 4};
    int m=3, n = 4, k1 = 1, k2 = -1;
    array double complex d[min(m, n)];
    array double complex d1[min(m, n-k1)];
    array double complex d2[min(m+k2, n)];

    d = cdiagonal(a);
    printf("cdiagonal(a) = %0.2f\n", d);

    d1 = cdiagonal(a, k1);
    printf("cdiagonal(a, 1) = %0.2f\n", d1);

    d2 = cdiagonal(a, k2);
    printf("cdiagonal(a, -1) = %0.2f\n", d2);
}
```

Output3

```
cdiagonal(a) = complex(1.00,2.00) complex(6.00,0.00) complex(4.00,0.00)

cdiagonal(a, 1) = complex(2.00,0.00) complex(7.00,0.00) complex(4.00,0.00)

cdiagonal(a, -1) = complex(5.00,0.00) complex(4.00,0.00)
```

See Also
diagonal(), **cdiagonalmatrix**().

References

cdiagonalmatrix

Synopsis
#include <**numeric.h**>
array double complex cdiagonalmatrix(**array double complex** *v*[&],... /* [int k] */)[:][:];

Syntax
cdiagonalmatrix(*v*)
cdiagonalmatrix(*v, k*)

Purpose
Form a complex diagonal matrix.

Return Value
This function returns a complex square matrix with specified diagonal elements.

Parameters
v An input complex vector.

k An input integer indicating specified diagonal elements of the matrix.

Description
The function **cdiagonalmatrix()** returns a complex square matrix of order $n + abs(k)$, with the elements of *v* on the the *k*th diagonal. $k = 0$ represents the main diagonal, $k > 0$ above the main diagonal, and $k < 0$ below the main diagonal.

Example

```
#include <numeric.h>
int main() {
    array double complex v[3] = {1,2,3};
    int n = 3, k;
    array double complex a[n][n];

    a = cdiagonalmatrix(v);
    printf("cdiagonal matrix a =\n%f\n", a);

    k = 0;
    a = cdiagonalmatrix(v,k);
    printf("cdiagonalmatrix(a, 0) =\n%f\n", a);

    k = 1;
    array double a2[n+abs(k)][n+abs(k)];
    a2 = cdiagonalmatrix(v,k);
    printf("cdiagonalmatrix(a2, 1) =\n%f\n", a2);

    k = -1;
    array double a3[n+abs(k)][n+abs(k)];
    a3 = cdiagonalmatrix(v,k);
```

```
    printf("cdiagonalmatrix(a3, -1) =\n%f\n", a3);
}
```

Output

```
cdiagonal matrix a =
complex(1.000000,0.000000) complex(0.000000,0.000000) complex(0.000000,0.000000)

complex(0.000000,0.000000) complex(2.000000,0.000000) complex(0.000000,0.000000)

complex(0.000000,0.000000) complex(0.000000,0.000000) complex(3.000000,0.000000)


cdiagonalmatrix(a, 0) =
complex(1.000000,0.000000) complex(0.000000,0.000000) complex(0.000000,0.000000)

complex(0.000000,0.000000) complex(2.000000,0.000000) complex(0.000000,0.000000)

complex(0.000000,0.000000) complex(0.000000,0.000000) complex(3.000000,0.000000)


cdiagonalmatrix(a2, 1) =
0.000000 1.000000 0.000000 0.000000
0.000000 0.000000 2.000000 0.000000
0.000000 0.000000 0.000000 3.000000
0.000000 0.000000 0.000000 0.000000

cdiagonalmatrix(a3, -1) =
0.000000 0.000000 0.000000 0.000000
1.000000 0.000000 0.000000 0.000000
0.000000 2.000000 0.000000 0.000000
0.000000 0.000000 3.000000 0.000000
```

See Also

diagonalmatrix(), **diagonal**().

References

cfevalarray

Synopsis
#include <**numeric.h**>
int cfevalarray(**array double** &*y*, **double complex** (**func*)(**double complex** *x*), **array double** &*x*, ...
/* [**array int** &*mask*, **double complex** *value*] */);

Syntax
cfevalarray(*y, func, x*)
cfevalarray(*y, func, x, mask, value*)

Purpose
Complex array function evaluation.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
y A **complex** output array which contains calculated function values.

x A **complex** input array at which the function values will be evaluated.

func A function routine given by the user.

mask An input array of integer. An entry with 1 indicates the corresponding function value will be evaluated, and an entry with 0 indicates that its output value is given by argument *value*.

value The default **complex** value which is used to replace the function value when the entry of *mask* is 0.

Description
This function evaluates the value of a user given function corresponding to the input array *x*. The input array can be of any data type and any dimension. The given function prototype shall be **double complex** *func*(**double complex**).

Example
Evaluation of function $f(x) = x^2$ for complex array of different dimensions.

```
#include <numeric.h>
#define N 2
#define M 3

double complex func(double complex x) {
   return x*x;
}

int main() {
  array float complex y2[N][M], x2[N][M]; /* diff data types, diff dim */
```

```
  array int mask1[N][M] = {1, 0, 1, 0, 1, 1};

  linspace(x2, 1, N*M);
  x2[1][2]=complex(2,3);
  cfevalarray(y2, func, x2);
  printf("y2 = %5.2f\n", y2);

  cfevalarray(y2, func, x2, mask1, -1);
  printf("y2 = %5.2f\n", y2);
}
```

Output

```
y2 = complex( 1.00, 0.00) complex( 4.00, 0.00) complex( 9.00, 0.00)
complex(16.00, 0.00) complex(25.00, 0.00) complex(-5.00,12.00)

y2 = complex( 1.00, 0.00) complex(-1.00, 0.00) complex( 9.00, 0.00)
complex(-1.00, 0.00) complex(25.00, 0.00) complex(-5.00,12.00)
```

See Also

fevalarray().

cfunm

Synopsis
#include <**numeric.h**>
int cfunm(**array double complex** *y[&][&]*, **double complex** (**func*)(**double complex**),
array double complex *x[&][&]*);

Syntax
cfunm(*y*, *func*, *x*)

Purpose
Evaluate general complex matrix function.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Input square matrix. It contains data to be evaluated.

func A function routine given by the user.

y Output square matrix which contains data of the calculated function values.

Description
This function evaluates the complex matrix version of the function specified by the parameter *func*. In this function, The input matrix *x* shall be **double complex** data type and the specified function prototype shall be **double complex** *func*(**double complex**). The output matrix *y* could be a **real** or **complex** data type as required.

Example
A complex matrix evaluation.

```
#include <numeric.h>
double complex mylog(double complex x) {
   return log(x);
}

int main() {
   array double complex zx[3][3]={complex(1,1),complex(2,2),0,
                          3,complex(4,1),complex(2,5),
                          0,0,0};
   array double complex zy[3][3];

   expm(zy,zx);
   printf("zx = \n%5.1f",zx);
   printf("zy = \n%5.1f",zy);
   cfunm(zx,mylog,zy);
```

```
   printf("zx = \n%5.1f",zx);
}
```

Output

```
zx =
complex(  1.0,  1.0) complex(  2.0,  2.0) complex(  0.0,  0.0)
complex(  3.0,  0.0) complex(  4.0,  1.0) complex(  2.0,  5.0)
complex(  0.0,  0.0) complex(  0.0,  0.0) complex(  0.0,  0.0)
zy =
complex(-44.9, 56.8) complex(-87.5, 70.9) complex(-101.5,-18.9)
complex(-12.5,118.7) complex(-57.4,175.5) complex(-155.5, 67.8)
complex(  0.0,  0.0) complex(  0.0,  0.0) complex(  1.0,  0.0)
zx =
complex(  1.0,  1.0) complex(  2.0,  2.0) complex(  0.0,  0.0)
complex(  3.0,  0.0) complex(  4.0,  1.0) complex(  2.0,  5.0)
complex(  0.0,  0.0) complex(  0.0,  0.0) complex(  0.0,  0.0)
```

See Also

cfunm(), **expm**(), **logm**(), **funm**(), **sqrtm**().

References

G. H. Golub, C. F. Van Loan, Matrix Computations Third edition, The Johns Hopkins University Press, 1996

charpolycoef

Synopsis
#include <**numeric.h**>
int charpolycoef(array double complex *p*[&], **array double complex** *a*[&][&]);

Purpose
Calculates the coefficients of characteristic polynomial of a matrix.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
p An output array which contains the calculated coefficients of the characteristic polynomial of a matrix.

a An input square matrix.

Description
This function calculates the coefficients of the characteristic polynomial of matrix **A** which is defined as $det(x\mathbf{I} - \mathbf{A})$. The coefficients are in the order of descending powers. The polynomial it represents is $p_0x^n + ... + p_{n-1}x + p_n$.

Example
This example calculates the polynomial coefficients for three different matrices *a1*, *a2* and *a3*. *a1* is symmetrical with real eigenvalues, *a2* is non-symmetrical with real eigenvalues, *a3* is non-symmetrical with complex eigenvalues, and *a4* is a complex matrix.

```
#include <numeric.h>
int main() {
     /* eigenvalues of a symmtrical matrix are always real numbers */
     array double a1[3][3] = {0.8, 0.2, 0.1,
                              0.2, 0.7, 0.3,
                              0.1, 0.3, 0.6};
     /* eigenvalues of a non-symmtrical matrix can be either real numbers or
        complex numbers */
     /* this non-symmtrical matrix has real eigenvalues */
     array double a2[3][3] = {0.8, 0.2, 0.1,
                             0.1, 0.7, 0.3,
                             0.1, 0.1, 0.6};
     /* this non-symmtrical matrix has complex eigenvalues */
     array double a3[3][3] = {3, 9, 23,
                              2, 2, 1,
                              -7, 1, -9};
    array double complex a4[2][2] = {complex(1,2), 4,
                                     3, 7};



     array double charpolynomial[4];
     array double complex zcharpolynomial[4];
     array double complex zcharpolynomial2[3];
```

```
    charpolycoef(charpolynomial, a1);
    printf("charpolynomial from charpolycoef(a1) =\n%f\n", charpolynomial);

    charpolycoef(charpolynomial, a2);
    printf("charpolynomial from charpolycoef(a2) =\n%f\n", charpolynomial);

    charpolycoef(charpolynomial, a3);
    printf("charpolynomial from charpolycoef(a3) =\n%f\n", charpolynomial);

    charpolycoef(zcharpolynomial, a3);
    printf("zcharpolynomial from charpolycoef(zcharpolynomial, a3) =\n%f\n",
            zcharpolynomial);

    charpolycoef(zcharpolynomial2, a4);
    printf("zcharpolynomial2 from charpolycoef(zcharpolynomial2, a4) =\n%f\n",
           zcharpolynomial2);
}
```

Output

```
charpolynomial from charpolycoef(a1) =
1.000000 -2.100000 1.320000 -0.245000

charpolynomial from charpolycoef(a2) =
1.000000 -2.100000 1.400000 -0.300000

charpolynomial from charpolycoef(a3) =
1.000000 4.000000 103.000000 -410.000000

zcharpolynomial from charpolycoef(zcharpolynomial, a3) =
complex(1.000000,0.000000) complex(4.000000,0.000000) complex(103.000000,0.000000)
complex(-410.000000,0.000000)

zcharpolynomial2 from charpolycoef(zcharpolynomial2, a4) =
complex(1.000000,0.000000) complex(-8.000000,-2.000000) complex(-5.000000,14.000000)
```

See Also

roots(), **polycoef**().

References

choldecomp

Synopsis
#include <**numeric.h**>
int choldecomp(**array double complex** *a*[&][&], **array double complex** *l*[&][&], ...
/* [**char** *mode*] */);

Syntax
choldecomp(*a, l*); choldecomp(*a, l, "mode"*);
Purpose
Computes the Cholesky factorization of a symmetric ,positive ,definite matrix A.

Return Value
This function returns 0 on success, a negative value on failure, and positive value i indicates that the leading minor of order i is not positive definite and the factorization could not be completed.

Parameters
a A symmetric positive definite two-dimensional matrix.

l A two-dimensional output matrix which contains the upper or lower triangle of the symmetric matrix *a*.

mode The it mode specifies the calculation of upper or lower triangle matrix.
'L' or 'l'- lower triangle is calculated.
otherwise the upper triangle matrix is calculated. By default, the upper triangle matrix is calculated.

Description
This function computes Cholesky factorization of a symmetric positive definite matrix **A**. Cholesky decomposition factors a symmetric positive definite matrix into two matrices. For a symmetric positive finite matrix **A** of real type, Cholesky decomposition can produce matrix **L** so that

$$\mathbf{A} = \mathbf{L^T L}$$

for upper trangle factorization or

$$\mathbf{A} = \mathbf{L L^T}$$

for lower triangle factorization. where L^T is the transpose of matrix **L**. For a symmetric positive finite matrix **A** of complex type, instead of L^T, the Hermitian L^H of matrix **L** shall be used.
Example1
Cholesky factorization of a real matrix.

```
#include <numeric.h>
int main() {
     int m = 5;
     array double a[5][5] = { 1, 1, 1, 1, 1,
                              1, 2, 3, 4, 5,
                              1, 3, 6,10,15,
                              1, 4, 10,20,35,
```

```
                              1, 5, 15,35,70};
     array double a1[5][5] = { -1, 1, 1, 1, 1,
                              1, 2, 3, 4, 5,
                              1, 3, 6,10,15,
                              1, 4, 10,20,35,
                              1, 5, 15,35,70};
     int status;
     array double l[m][m];

     status = choldecomp(a, l);
     if (status == 0) {
        printf("upper triangle l =\n%f\n", l);
        printf("transpose(l)*l - a =\n%f\n", transpose(l)*l-a);
     }
     else
        printf("error: Matrix must be positive definite.\n"
                "The %d order of matrix is not positive definite.\n",status);

     status = choldecomp(a, l, 'l');
     if (status == 0) {
        printf("lower triangle l =\n%f\n", l);
        printf("l*transpose(l) - a =\n%f\n", l*transpose(l)-a);
     }
     else
        printf("error: Matrix must be positive definite.\n"
                "The %d order of matrix is not positive definite.\n",status);

     status = choldecomp(a1, l);
     if (status == 0) {
        printf("l =\n%f\n", l);
     }
     else {
        printf("a =\n%f", a1);
        printf("error: Matrix must be positive definite.\n"
                "The %d order of matrix is not positive definite.\n",status);
     }
}
```

Output1

```
upper triangle l =
1.000000 1.000000 1.000000 1.000000 1.000000
0.000000 1.000000 2.000000 3.000000 4.000000
0.000000 0.000000 1.000000 3.000000 6.000000
0.000000 0.000000 0.000000 1.000000 4.000000
0.000000 0.000000 0.000000 0.000000 1.000000

transpose(l)*l - a =
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000

lower triangle l =
1.000000 0.000000 0.000000 0.000000 0.000000
1.000000 1.000000 0.000000 0.000000 0.000000
1.000000 2.000000 1.000000 0.000000 0.000000
1.000000 3.000000 3.000000 1.000000 0.000000
```

```
1.000000 4.000000 6.000000 4.000000 1.000000

l*transpose(l) - a =
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000

a =
-1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 3.000000 6.000000 10.000000 15.000000
1.000000 4.000000 10.000000 20.000000 35.000000
1.000000 5.000000 15.000000 35.000000 70.000000
error: Matrix must be positive definite.
The 1 order of matrix is not positive definite.
```

Example2
Cholesky factorization of a complex matrix.

```
#include <numeric.h>

int main() {
     int m = 3;
     array double complex a[3][3] =
           { complex(2,0), complex(0,-1), complex(0,0),
             complex(0,1), complex(2,0), complex(0,0),
             complex(0,0), complex(0,0), complex(3,0)};

     int status;
     array double complex l[m][m];

     status = choldecomp(a, l);
     if (status == 0) {
        printf("upper triangle l =\n%5.3f\n", l);
        printf("transpose(l)*l - a =\n%5.3f\n", conj(transpose(l))*l-a);
     }
     else
        printf("error: Matrix must be positive definite.\n"
                "The %d order of matrix is not positive definite.\n",status);

     status = choldecomp(a, l, 'l');
     if (status == 0) {
        printf("lower triangle l =\n%5.3f\n", l);
        printf("l*transpose(l) - a =\n%5.3f\n", l*conj(transpose(l))-a);
     }
     else
        printf("error: Matrix must be positive definite.\n"
                "The %d order of matrix is not positive definite.\n",status);

}
```

Output2

```
upper triangle l =
complex(1.414,0.000) complex(0.000,-0.707) complex(0.000,0.000)
complex(0.000,0.000) complex(1.225,0.000) complex(0.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(1.732,0.000)
```

```
transpose(l)*l - a =
complex(0.000,0.000) complex(0.000,0.000) complex(0.000,0.000)
complex(0.000,0.000) complex(-0.000,0.000) complex(0.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(-0.000,0.000)

lower triangle l =
complex(1.414,0.000) complex(0.000,0.000) complex(0.000,0.000)
complex(0.000,0.707) complex(1.225,0.000) complex(0.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(1.732,0.000)

l*transpose(l) - a =
complex(0.000,0.000) complex(0.000,0.000) complex(0.000,0.000)
complex(0.000,0.000) complex(-0.000,0.000) complex(0.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(-0.000,0.000)
```

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

cinverse

Synopsis
#include <**numeric.h**>
array double complex cinverse(**array double complex** *a*[:][:], ... /* [**int** **status*] */)[:][:];

Syntax
cinverse(*a*)
cinverse(*a*, *status*)

Purpose
Calculate the inverse of a complex square matrix.

Return Value
This function returns the inverse matrix.

Parameters
a Input square matrix of double complex type.

status Output integer indicating the status of calculation.

Description
This function calculates the inverse matrix of a complex square matrix. If calculation is successful,$status = 0$, otherwise $status \neq 0$.

Example

```
#include <numeric.h>
int main() {
    array double complex a[2][2] = {complex(2,-3), 4,
                                                3, 7}; // n-by-n matrix
    array double complex inv[2][2];
    int status;

    inv = cinverse(a);
    printf("cinverse(a) =\n%f\n", inv);
    printf("a =\n%f\n", a);
    printf("cinverse(a)*a =\n%f\n", inv*a);
    printf("a*cinverse(a) =\n%f\n", a*inv);

    inv = cinverse(a, &status);
    if(status == 0)
        printf("cinverse(a, &status) = %f\n", inv);
    else
        printf("error: numerical error in cinverse()\n");
}
```

Output

```
cinverse(a) =
complex(0.031461,0.330337) complex(-0.017978,-0.188764)
complex(-0.013483,-0.141573) complex(0.150562,0.080899)

a =
complex(2.000000,-3.000000) complex(4.000000,0.000000)
complex(3.000000,0.000000) complex(7.000000,0.000000)

cinverse(a)*a =
complex(1.000000,-0.000000) complex(0.000000,0.000000)
complex(-0.000000,-0.000000) complex(1.000000,0.000000)

a*cinverse(a) =
complex(1.000000,-0.000000) complex(0.000000,0.000000)
complex(0.000000,-0.000000) complex(1.000000,0.000000)

cinverse(a, &status) = complex(0.031461,0.330337) complex(-0.017978,-0.188764)
complex(-0.013483,-0.141573) complex(0.150562,0.080899)
```

See Also

inverse(), **ludecomp**(), **clinsolve**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadephia, 1992.

clinsolve

Synopsis
#include <numeric.h>
int clinsolve(array double complex *x*[:], **array double complex** *a*[:][:], **array double complex** *b*[:]);

Purpose
Solve the linear system of equations by LU decomposition.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Output array which contains the solution of equations.

a Input two-dimensional array contains the coefficients of a linear system of equations.

b Input one-dimensional array for linear equations.

Description
This function solves the system of complex linear equations by LU decomposition. The input matrix *a* should be n-by-n. The function can handle the equation set with complex numbers. The function returns 0 if there is a solution for the equation set, otherwise it returns -1.

Algorithm
This function uses the subroutines **zgesv**() from LAPACK. It calculates the solution to a real/complex system of linear equations $a * X = b$, where a is an n-by-n matrix. The LU decomposition with partial pivoting and row interchanges is used to factor *a* as $a = P * L * U$, where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of *a* is then used to solve the system of equation $a * X = b$.

Example
Solution of a linear system of equations with complex coefficients.

```
#include <numeric.h>
int main() {
     array double complex a[3][3] = {complex(3,4), 0, 6,
                              0, 2, 1,
                              1, 0, 1};
     array double complex b[3] = {2,
                                  13,
                                  25};
     array double complex x[3];
     int status;

     clinsolve(x, a, b);
     printf("linsolve(x, a,b) =\n%f\n", x);
```

```
    status = clinsolve(x, a, b);
    if(status == 0)
        printf("clinsolve(x,a,b) =\n%f\n", x);
    else
        printf("error: numerical error in clinsolve()\n");
}
```

Output

```
linsolve(x, a,b) =
complex(17.760000,23.680000) complex(2.880000,11.840000) complex(7.240000,-23.680000)


clinsolve(x,a,b) =
complex(17.760000,23.680000) complex(2.880000,11.840000) complex(7.240000,-23.680000)
```

See Also

llsqsolve(), **linsolve**(), **inverse**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

cmean

Synopsis
#include <**numeric.h**>
double complex cmean(**array double complex** &*a*, ... /*[**array double complex** *v[&]*] */);

Syntax
cmean(*a*)
cmean(*a*, *v*)

Purpose
Calculate the mean value of all elements of an array of complex type and mean values of the elements of each row of a two-dimensional array of complex type.

Return Value
This function returns the mean value of all the elements of complex type.

Parameters
a An input array of any dimension.

v An output array which contains the mean values of each row of a two-dimensional array of complex type. *v* shall be an array of double complex type.

Description
This function calculates the complex mean value of all elements in an array of any dimension. If the array is a two-dimensional matrix, the function can calculate complex mean values of each row. The complex mean values of each row are passed out by argument *v*.

Example
Calculate the mean values of all elements of arrays of complex type with different dimensions. The mean values of each row are passed by argument *v*.

```
#include <numeric.h>
#define N 2
#define M 3

int main() {
    complex a[3] = {1,2,complex(3.435,6.35765)};
    double complex c[2][3][5];
    c[0][0][0] = complex(53.327,92.5310);
    c[0][0][1] = 20;
    complex a1[N][M] = {1,complex(3.54,54.43),3,
                        4,5,complex(3.455,466)};
    array double complex b1[3][4] = {1,2,3,complex(4,64.54),
                                     complex(5,6.45),6,7,8,
                                     1,complex(4.35,54.43),3,4};
    array double complex meana1[N], meanb1[3];
```

```
    double complex meanval;

    meanval = cmean(a);
    printf("mean(a) = %f\n", meanval);
    meanval = cmean(c);
    printf("mean(c) = %f\n", meanval);

    /* Note: second argument of cmean() must be double complex data type */

    meanval = cmean(a1, meana1);
    printf("meanval = mean(a1, meana1) = %f\n", meanval);
    printf("mean(a1, meana1) = %f\n", meana1);

    meanval = cmean(b1, meanb1);
    printf("meanval = mean(b1, meanb1) = %f\n", meanval);
    printf("mean(b1, meanb1) = %0.2f\n", meanb1);
}
```

Output

```
mean(a) = complex(2.145000,2.119217)
mean(c) = complex(2.444233,3.084367)
meanval = mean(a1, meana1) = complex(3.332500,86.738333)
mean(a1, meana1) = complex(2.513333,18.143333) complex(4.151667,155.333333)

meanval = mean(b1, meanb1) = complex(4.029167,10.451667)
mean(b1, meanb1) = complex(2.50,16.14) complex(6.50,1.61) complex(3.09,13.61)
```

See Also

mean().

combination

Synopsis
#include <**numeric.h**>
unsigned long long combination(**unsigned int** *n*, **unsigned int** *k*);

Purpose
Calculate the number of combinations of *n* different things taken *k* at a time without repetitions.

Return Value
This function returns the number of combinations.

Parameters
n The number of different things.

k The number of items taken from *n* different things.

Description
This function calculates the number of combination of *n* different things taken *k* at a time without repetitions. The number of combination is the number of sets that can be made up from *n* things, each set containing *k* different things and no two sets containing exactly the same *k* things.

Algorithm
The number of combination of *n* different things taken *k* at a time without repetitions is defined as

$$C_k^n = \frac{n!}{(n-k)!k!}$$

Example

```
#include <numeric.h>
int main() {
    unsigned long long f;

    f = combination(3, 2);
    printf("combination(3, 2) = %llu\n", combination(3, 2));
}
```

Output

```
combination(3, 2) = 3
```

See Also
factorial().

References

companionmatrix

Synopsis
#include <**numeric.h**>
array double companionmatrix(**array double** *v*[&])[:][:];

Purpose
Find a companion matrix.

Return Value
This function returns a companion matrix.

Parameters
v Input array containing the coefficients of a polynomial.

Description
This function returns the corresponding companion matrix of array *v* which contains the coefficients of a polynomial. The eigenvalues of companion matrix are roots of the polynomial.

Example

```
#include <numeric.h>
int main() {
    array double v[4] = {2,3,4,5}; /* 2x^3 + 3x^2 + 4x + 5 */
    int n = 4;
    array double a[n-1][n-1];

    a = companionmatrix(v);
    printf("companionmatrix(a) =\n%f\n", a);
}
```

Output

```
companionmatrix(a) =
-1.500000 -2.000000 -2.500000
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
```

See Also
ccompanionmatrix(), **eigensystem**(), **polycoef**().

References

complexsolve

Synopsis
#include <**numeric.h**>
int complexsolve(int *n1*, **int** *n2*, **double** *phi_or_r1*, **double** *phi_or_r2*, **double complex** *z3*,
double &*xx1*, **double** &*xx2*, **double** &*xx3*, **double** &*xx4*);

Purpose
Solve a complex equation in the polar format.

Return Value
This function returns the number of solutions with 0, 1, or 2.

Parameters
n1 The first position of the two unknowns on the left hand side of equation(1). It's value can be 1, 2, 3, or 4.

n2 The second position of the two unknowns on the left hand side of equation(1). It's value can be 1, 2, 3, or 4.

phi_or_r1 The value of the first known on the left hand side.

phi_or_r2 The value of the second known on the left hand side.

z3 Complex number on the right hand side.

xx1 First unknown on the left hand side.

xx2 Second unknown on the left hand side.

xx3 First unknown on the left hand side for the second solution, if there are more than one set of solution.

xx4 Second unknown on the left hand side for the second solution, if there are more than one set of solution.

Description
This function is used to solve a complex equation in polar format. The equation is in the form of

$$R_1 e^{i\phi_1} + R_2 e^{i\phi_2} = x_3 + iy_3 \tag{11.1}$$

Because it is a complex equation that can be partitioned into real and imaginary parts, two unknowns out of four parameters R_1, ϕ_1, R_2, and ϕ_2 can be solved in this equation. The parameters R_1, ϕ_1, R_2, and ϕ_2 are in positions 1, 2, 3, and 4, respectively.

Algorithm
Two unknowns can be solved by decomposing equation (11.1) into real and imaginary parts

$$R_1 \cos\phi_1 + R_2 \cos\phi_2 = x_3 \tag{11.2}$$

$$R_1 \sin\phi_1 + R_2 \sin\phi_2 = y_3 \tag{11.3}$$

Case 1: $n_1 = 1, n_2 = 2$, Solve for R_1 and ϕ_1, given R_2, ϕ_2, R_3, ϕ_3 or R_2, ϕ_2, x_3, y_3. Where, n_1 and n_2 are the first and second positions of two unknowns on the left hand side of equation (11.1), respectively.

From equations (11.2) and equations (11.3), we get

$$R_1 \cos\phi_1 = x_3 - R_2 \cos\phi_2 = a \tag{11.4}$$

$$R_1 \sin\phi_1 = y_3 - R_2 \sin\phi_2 = b \tag{11.5}$$

R_1 and ϕ_1 can be calculated as

$$\begin{aligned} R_1 &= \sqrt{a^2 + b^2} && (11.6)\\ \phi_1 &= \operatorname{atan2}(b, a) && (11.7) \end{aligned}$$

Case 2: $n_1 = 1, n_2 = 3$, Given ϕ_1 and ϕ_2, R_1 and R_2 are solved.

Multiplying equation (11.1) by $e^{-i\phi_2}$ and $e^{-i\phi_1}$ gives

$$R_1 e^{i(\phi_1 - \phi_2)} + R_2 = R_3 e^{i(\phi_3 - \phi_2)} \tag{11.8}$$

$$R_1 + R_2 e^{i(\phi_2 - \phi_1)} = R_3 e^{i(\phi_3 - \phi_1)} \tag{11.9}$$

Imaginary parts of equations (11.8) and equation (11.9) are

$$R_1 \sin(\phi_1 - \phi_2) = R_3 \sin(\phi_3 - \phi_2) \tag{11.10}$$

$$R_2 \sin(\phi_2 - \phi_1) = R_3 \sin(\phi_3 - \phi_1) \tag{11.11}$$

From equation (11.10) and equation (11.11) we get

$$R_1 = R_3 \frac{\sin(\phi_3 - \phi_2)}{\sin(\phi_1 - \phi_2)} \tag{11.12}$$

$$R_2 = R_3 \frac{\sin(\phi_3 - \phi_1)}{\sin(\phi_2 - \phi_1)} \tag{11.13}$$

Case 3: $n_1 = 1, n_2 = 4$, Given ϕ_1 and R_2, R_1 and ϕ_2 are solved.

From equation (11.2) we have

$$R_1 = \frac{x_3 - R_2 \cos\phi_2}{\cos\phi_1} \tag{11.14}$$

substitute equation (11.14) into equation (11.3) we get

$$(x_3 - R_2 \cos\phi_2) \sin\phi_1 + R_2 \sin\phi_2 \cos\phi_1 = y_3 \cos\phi_1 \tag{11.15}$$

equation (11.15) can be simplified as

$$\sin(\phi_2 - \phi_1) = \frac{y_3 \cos\phi_1 - x_3 \sin\phi_1}{R_2} = a \tag{11.16}$$

then

$$\phi_2 = \phi_1 + \sin^{-1}(a)\phi_2 = \phi_1 + \pi - \sin^{-1}(a) \tag{11.17}$$

If $\cos\phi_1$ is larger than machine epsilon,ε, R_1 can be obtained using equation (11.14). Otherwise R_1 will be obtained using equation (11.3)
if ($|\cos\phi_1)| >$ FLT_EPSILON

$$R_1 = \frac{(x_3 - R_2\cos\phi_2)}{\cos\phi_1} \tag{11.18}$$

else

$$R_1 = \frac{(y_3 - R_2\sin\phi_2)}{\sin\phi_1} \tag{11.19}$$

Case 4: $n_1 = 2, n_2 = 4$, given R_1 and R_2, ϕ_1 and ϕ_2 in equation (11.1) can be solved.

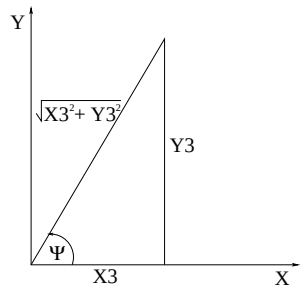
From equation (11.1), we get

$$\cos\phi_1 = \frac{x_3 - R_2\cos\phi_2}{R_1}, \quad \sin\phi_1 = \frac{y_3 - R_2\sin\phi_2}{R_1} \tag{11.20}$$

Substituting these results into the identity equation $\sin^2\phi_1 + \cos^2\phi_1 = 1$ and simplifying the resultant equation, we get

$$y_3\sin\phi_2 + x_3\cos\phi_2 = \frac{x_3^2 + y_3^2 + R_2^2 - R_1^2}{2R_2} \tag{11.21}$$

from equation (11.21), we can derive the formulas for ϕ_1 and ϕ_2 as follows:



$$\tan(\psi) = \frac{y_3}{x_3}, \cos(\psi) = \frac{x_3}{\sqrt{x_3^2 + y_3^2}}, \sin(\psi) = \frac{y_3}{\sqrt{x_3^2 + y_3^2}}$$

Equation (11.21) becomes,

$$\frac{y_3}{\sqrt{x_3^2 + y_3^2}}\sin(\phi_2) + \frac{x_3}{\sqrt{x_3^2 + y_3^2}}\cos(\phi_2) = \frac{x_3^2 + y_3^2 + R_2^2 - R_1^2}{2R_2\sqrt{x_3^2 + y_3^2}} \tag{11.22}$$

$$a = \sin(\psi)\sin(\phi_2) + \cos(\psi)\cos(\phi_2)$$

Let,

$$a = \frac{x_3^2 + y_3^2 + R_2^2 - R_1^2}{2R_2\sqrt{x_3^2 + y_3^2}} \tag{11.23}$$

Becomes,

$$\cos(\phi_2 - \psi) = a \tag{11.24}$$

$$\begin{aligned} \phi_2 &= \psi \pm \cos^{-1}(a) && (11.25)\\ &= \operatorname{atan2}(y_3, x_3) \pm \operatorname{acos}\left(\frac{x_3^2 + y_3^2 + R_2^2 - R_1^2}{2R_2\sqrt{x_3^2 + y_3^2}}\right) && (11.26) \end{aligned}$$

ϕ_1 can be obtained using equation (11.20)

$$\phi_1 = \operatorname{atan2}(\sin\phi_1, \cos\phi_1) \tag{11.27}$$

or use identity equation,

$$\tan(\frac{\phi}{2}) = \frac{1-\cos(\phi)}{\sin(\phi)} \text{ or } \tan(\frac{\phi}{2}) = \frac{\sin(\phi)}{1+\cos(\phi)} \tag{11.28}$$

$$\phi = 2\tan^{-1}(\frac{1-\cos(\phi)}{\sin(\phi)}) \text{ or } \phi = 2\tan^{-1}(\frac{\sin(\phi)}{1+\cos(\phi)} \tag{11.29}$$

Case 5: $n_1 = 2, n_2 = 3$, given R_1 and ϕ_2, ϕ_1 and R_2 are solved. This case is similar to Case 3.

Case 6: $n_1 = 3, n_2 = 4$, given R_1 and ϕ_1, R_2 and ϕ_2 are solved. This case is similar to case 1.

Example 1

A complex equation

$$R_1 e^{i\phi_1} + R_2 e^{i\phi_2} = 1 + i2$$

is solved for the 6 cases described above.

```
#include <numeric.h>
#include <complex.h>

int main() {
    double x1, x2, x3, x4;
    int n1, n2, num;
    double phi_or_r1, phi_or_r2;
    complex z3, error;

    phi_or_r1 = 3;
    phi_or_r2 = 4;
    z3 = complex(1,2);

    /************* test n1 = 1, n2 = 2 ***********/
    n1 = 1,
    n2 = 2;
    num = complexsolve(n1, n2, phi_or_r1, phi_or_r2, z3, x1, x2, x3, x4);
    error = polar(x1, x2)  + polar(phi_or_r1, phi_or_r2) - z3;
    printf("For n1=1, n2=2, num = %d\n", num);
```

```
    printf("For n1=1, n2=2, the output x is %f %f %f %f  \n", x1, x2, x3, x4);
    printf("the residual error is %f \n", error);

    /************* test n1 = 1, n2 = 3 ***********/
    n1 = 1,
    n2 = 3;
    num = complexsolve(n1, n2, phi_or_r1, phi_or_r2, z3, x1, x2, x3, x4);
    printf("For n1=1, n2=3, num = %d\n", num);
    printf("For n1=1, n2=3, the output x is %f %f %f %f  \n", x1, x2, x3, x4);

    /************* test n1 = 1, n2 = 4 ***********/
    n1 = 1,
    n2 = 4;
    num = complexsolve(n1, n2, phi_or_r1, phi_or_r2, z3, x1, x2, x3, x4);
    printf("For n1=1, n2=4, num = %d\n", num);
    printf("For n1=1, n2=4, the output x is %f %f %f %f  \n", x1, x2, x3, x4);

    /************* test n1 = 2, n2 = 3 ***********/
    n1 = 2,
    n2 = 3;
    num = complexsolve(n1, n2, phi_or_r1, phi_or_r2, z3, x1, x2, x3, x4);
    printf("For n1=2, n2=3, num = %d\n", num);
    printf("For n1=2, n2=3, the output x is %f %f %f %f  \n", x1, x2, x3, x4);

    /************* test n1 = 2, n2 = 4 ***********/
    n1 = 2,
    n2 = 4;
    num = complexsolve(n1, n2, phi_or_r1, phi_or_r2, z3, x1, x2, x3, x4);
    printf("For n1=2, n2=4, num = %d\n", num);
    printf("For n1=2, n2=4, the output x is %f %f %f %f  \n", x1, x2, x3, x4);

    /************* test n1 = 3, n2 = 4 ***********/
    n1 = 3,
    n2 = 4;
    num = complexsolve(n1, n2, phi_or_r1, phi_or_r2, z3, x1, x2, x3, x4);
    printf("For n1=3, n2=4, num = %d\n", num);
    printf("For n1=3, n2=4, the output x is %f %f %f %f  \n", x1, x2, x3, x4);
}
```

Output

Example 2

For a complex equation

$$R_1 e^{i\phi_1} + R_2 e^{i\phi_2} = 1 + i2$$

(1) Given $R_1 = 3$ and $\phi_1 = 4$ radian, find R_2 and ϕ_2
(2) Given $R_1 = 3$ and $R_2 = 4$, find ϕ_1 and ϕ_2

```
#include <numeric.h>
#include <complex.h>

int main () {
  double x1, x2, x3, x4;
  int n1, n2, num;
  double phi_or_r1, phi_or_r2;
  complex z3, error;

  /* problem (1) n1 = 3, n2 = 4 */
```

```
  phi_or_r1 = 3;
  phi_or_r2 = 4;
  z3 = complex(1,2);
  n1 = 3,
  n2 = 4;
  num = complexsolve(n1, n2, phi_or_r1, phi_or_r2, z3, x1, x2, x3, x4);
  printf("For n1=3, n2=4, the number of solution is = %d\n", num);
  printf("R2 = %f phi2 = %f\n", x1, x2);
  printf("R2 = %f phi2 = %f\n", x3, x4);

  /* problem (2), n1 = 2, n2 = 4 */
  n1 = 2,
  n2 = 4;
  num = complexsolve(n1, n2, phi_or_r1, phi_or_r2, z3, x1, x2, x3, x4);
  printf("For n1=2, n2=4, the number of solution is = %d\n", num);
  printf("phi1 = %f phi2 = %f\n", x1, x2);
  printf("phi1 = %f phi2 = %f\n", x3, x4);
}
```

Output

```
For n1=3, n2=4, the number of solution is = 1
R2 = 5.196488 phi2 = 0.964540
R2 = NaN phi2 = NaN
For n1=2, n2=4, the number of solution is = 2
phi1 = -0.613277 phi2 = 1.942631
phi1 = 2.827574 phi2 = 0.271667
```

See Also

complexsolvePP(), **complexsolvePR**(), **complexsolveRP**(), **complexsolveRR**(), **complexsolveRRz**(), **polar**().

References

complexsolvePP

Synopsis
#include <**numeric.h**>
int complexsolvePP(**double** *phi1*, **double** *phi2*, **double complex** *z3*,
double &*r1*, **double** &*r2*);

Purpose
Solve for r_1 and r_2 in a complex equation in the polar format.

Return Value
This function returns the number of solution with the value 1.

Parameters
phi1 The known value on the left hand side.

phi2 The known value on the left hand side.

z3 Complex number on the right hand side.

r1 r_1 for the solution on the left hand site.

r2 r_2 for the solution on the left hand site.

Description
This function is used to solve for r_1 and $r2_2$ in a complex equation in the polar format. The equation is in the form of

$$R_1 e^{i\phi_1} + R_2 e^{i\phi_2} = x_3 + iy_3 \tag{11.30}$$

Because it is a complex equation that can be partitioned into real and imaginary parts, two unknowns R_1 and ϕ_2 can be solved in this equation.

Algorithm
Two unknowns can be solved by decomposing equation (11.30) into real and imaginary parts

$$R_1 \cos\phi_1 + R_2 \cos\phi_2 = x_3 \tag{11.31}$$

$$R_1 \sin\phi_1 + R_2 \sin\phi_2 = y_3 \tag{11.32}$$

Multiplying equation (11.30) by $e^{-i\phi_2}$ and $e^{-i\phi_1}$ gives

$$R_1 e^{i(\phi_1-\phi_2)} + R_2 = R_3 e^{i(\phi_3-\phi_2)} \tag{11.33}$$

$$R_1 + R_2 e^{i(\phi_2-\phi_1)} = R_3 e^{i(\phi_3-\phi_1)} \tag{11.34}$$

Imaginary parts of equations (11.33) and equation (11.34) are

$$R_1 \sin(\phi_1 - \phi_2) = R_3 \sin(\phi_3 - \phi_2) \tag{11.35}$$

$$R_2 \sin(\phi_2 - \phi_1) = R_3 \sin(\phi_3 - \phi_1) \tag{11.36}$$

From equation (11.35) and equation (11.36) we get

$$R_1 = R_3 \frac{\sin(\phi_3 - \phi_2)}{\sin(\phi_1 - \phi_2)} \tag{11.37}$$

$$R_2 = R_3 \frac{\sin(\phi_3 - \phi_1)}{\sin(\phi_2 - \phi_1)} \tag{11.38}$$

Note

The function call

```
complexsolvePP(phi1, phi2, z3, r1, r2);
```

is equivalent to

```
complexsolve(1, 3, phi1, phi2, z3, r1, r2, 0, 0);
```

Example

For a complex equation

$$R_1 e^{i\phi_1} + R_2 e^{i\phi_2} = 1 + i2$$

Given $\phi_1 = 3$ radian and $\phi_2 = 4$ radian, find R_1 and ϕ_2.

See Also

complexsolve(), **complexsolvePR**(), **complexsolveRP**(), **complexsolveRR**(), **complexsolveRRz**(), **polar**().

References

complexsolvePR

Synopsis
#include <**numeric.h**>
int complexsolvePR(**double** *phi1*, **double** *r2*, **double complex** *z3*,
double &*r1*, **double** &*phi2*, **double** &*r1_2*, **double** &*phi2_2*);

Purpose
Solve for r_1 and ϕ_2 in a complex equation in the polar format.

Return Value
This function returns the number of solutions with the value 2.

Parameters
phi1 The known value on the left hand side.

r2 The known value on the left hand side.

z3 Complex number on the right hand side.

r1 r_1 for the first solution on the left hand site.

phi2 ϕ_2 for the first solution on the left hand site.

r1_2 r_1 for the second solution on the left hand site.

phi2_2 ϕ_2 for the second solution on the left hand site.

Description
This function is used to solve for r_1 and ϕ_2 in a complex equation in the polar format. The equation is in the form of

$$R_1 e^{i\phi_1} + R_2 e^{i\phi_2} = x_3 + iy_3 \tag{11.39}$$

Because it is a complex equation that can be partitioned into real and imaginary parts, two unknowns R_1 and ϕ_2 can be solved in this equation.

Algorithm
Two unknowns can be solved by decomposing equation (11.39) into real and imaginary parts

$$R_1 \cos\phi_1 + R_2 \cos\phi_2 = x_3 \tag{11.40}$$

$$R_1 \sin\phi_1 + R_2 \sin\phi_2 = y_3 \tag{11.41}$$

From equation (11.40) we have

$$R_1 = \frac{x_3 - R_2 \cos\phi_2}{\cos\phi_1} \tag{11.42}$$

substitute equation (11.42) into equation (11.41) we get

$$(x_3 - R_2\cos\phi_2)\sin\phi_1 + R_2\sin\phi_2\cos\phi_1 = y_3\cos\phi_1 \tag{11.43}$$

equation (11.43) can be simplified as

$$\sin(\phi_2 - \phi_1) = \frac{y_3\cos\phi_1 - x_3\sin\phi_1}{R_2} = a \tag{11.44}$$

then

$$\phi_2 = \phi_1 + \sin^{-1}(a)\phi_2 = \phi_1 + \pi - \sin^{-1}(a) \tag{11.45}$$

If $\cos\phi_1$ is larger than machine epsilon,ε, R_1 can be obtained using equation (11.42). Otherwise R_1 will be obtained using equation (11.41)
if ($|\cos\phi_1)| >$ FLT_EPSILON

$$R_1 = \frac{(x_3 - R_2\cos\phi_2)}{\cos\phi_1} \tag{11.46}$$

else

$$R_1 = \frac{(y_3 - R_2\sin\phi_2)}{\sin\phi_1} \tag{11.47}$$

Note
The function call

```
complexsolvePR(phi1, r2, z3, r1, phi2, r1_2, phi2_2);
```

is equivalent to

```
complexsolve(1, 4, phi1, r2, z3, r1, phi2, r1_2, phi2_2);
```

Example
For a complex equation

$$R_1 e^{i\phi_1} + R_2 e^{i\phi_2} = 1 + i2$$

Given $\phi_1 = 3$ radian and $r_2 = 4$ meters, find R_1 and ϕ_2.
See Also
complexsolve(), **complexsolvePP**(), **complexsolveRP**(), **complexsolveRR**(), **complexsolveRRz**(), **polar**().

References

complexsolveRP

Synopsis
#include <**numeric.h**>
int complexsolveRP(**double** *a*, **double complex** *z*,
double &*r*, **double** &*theta*, **double** &*r_2*, **double** &*theta_2*);

Purpose
Solve for r and θ in a complex equation in the polar format.

Return Value
This function returns the number of solutions with the value 2 if successful. Otherwise, it returns -1.

Parameters
a The known value on the left hand side.

z The complex number on the right hand side.

r r for the first solution on the left hand site.

theta θ for the first solution on the left hand site.

r_2 r for the second solution on the left hand site.

theta_2 θ for the second solution on the left hand site.

Description
This function is used to solve for r and θ in a complex equation in the polar format. The equation is in the form of

$$(a + ir)e^{i\theta} = x + iy \tag{11.48}$$

Because it is a complex equation that can be partitioned into real and imaginary parts, two unknowns r and θ can be solved in this equation.

Algorithm
Two unknowns can be solved by decomposing equation (11.48) into real and imaginary parts

$$a\cos\theta - r\sin\theta = x \tag{11.49}$$

$$a\sin\theta + r\cos\theta = y \tag{11.50}$$

From equations (11.49) and (11.50), we have

$$(a\cos\theta - r\sin\theta)^2 + (a\sin\theta + r\cos\theta)^2 = x^2 + y^2 \tag{11.51}$$

which can be simplified to

$$a^2 + r^2 = x^2 + y^2 \tag{11.52}$$

From equation (11.51), we can find two solutions for r as follows:

$$r = \pm\sqrt{x^2 + y^2 - a^2} \tag{11.53}$$

From equations (11.49) and (11.50), we can derive the formulas for $\sin\theta$ and $\cos\theta$ as follows:

$$\sin\theta = \frac{ay - rx}{r^2 + a^2} \tag{11.54}$$

$$\cos\theta = \frac{ry + ax}{r^2 + a^2} \tag{11.55}$$

For each r, we can use the function **atan2()** to find θ.

Example
For a complex equation

$$(a + ir)e^{i\theta} = 1 + i2$$

Given $a = 3$ meters and $r_2 = 4$ meters, find r and θ.
See Also
complexsolve(), **complexsolvePP**(), **complexsolvePR**(), **complexsolveRR**(), **complexsolveRRz**(), **polar**().

References

complexsolveRR

Synopsis
#include <**numeric.h**>
int complexsolveRR(**double** *r1*, **double** *r2*, **double complex** *z3*,
double &*phi1*, **double** &*phi2*, **double** &*phi1_2*, **double** &*phi2_2*);

Purpose
Solve for ϕ_1 and ϕ_2 in a complex equation in the polar format.

Return Value
This function returns the number of solutions with the value 2.

Parameters
r1 The known value on the left hand side.

r2 The knonw value on the left hand side.

z3 Complex number on the right hand side.

phi1 ϕ_1 for the first solution on the left hand side.

phi2 ϕ_2 for the first solution on the left hand side.

phi1_2 ϕ_1 for the second solution on the left hand side.

phi2_2 ϕ_2 for the second solution on the left hand side.

Description
This function is used to solve for ϕ_1 and ϕ_2 in a complex equation in polar format. The equation is in the form of

$$R_1 e^{i\phi_1} + R_2 e^{i\phi_2} = x_3 + iy_3 \tag{11.56}$$

Because it is a complex equation that can be partitioned into real and imaginary parts, two unknowns ϕ_1 and ϕ_2 can be solved in this equation.

Algorithm
Two unknowns can be solved by decomposing equation (11.56) into real and imaginary parts

$$R_1 \cos\phi_1 + R_2 \cos\phi_2 = x_3 \tag{11.57}$$

$$R_1 \sin\phi_1 + R_2 \sin\phi_2 = y_3 \tag{11.58}$$

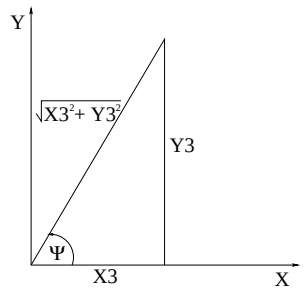
From equation (11.56), we get

$$\cos\phi_1 = \frac{x_3 - R_2\cos\phi_2}{R_1}, \quad \sin\phi_1 = \frac{y_3 - R_2\sin\phi_2}{R_1} \tag{11.59}$$

Substituting these results into the identity equation $\sin^2\phi_1 + \cos^2\phi_1 = 1$ and simplifying the resultant equation, we get

$$y_3 \sin\phi_2 + x_3 \cos\phi_2 = \frac{x_3^2 + y_3^2 + R_2^2 - R_1^2}{2R_2} \tag{11.60}$$

from equation (11.60), we can derive the formulas for ϕ_1 and ϕ_2 as follows:



$$\tan(\psi) = \frac{y_3}{x_3}, \cos(\psi) = \frac{x_3}{\sqrt{x_3^2 + y_3^2}}, \sin(\psi) = \frac{y_3}{\sqrt{x_3^2 + y_3^2}}$$

Equation (11.60) becomes,

$$\frac{y_3}{\sqrt{x_3^2 + y_3^2}} \sin(\phi_2) + \frac{x_3}{\sqrt{x_3^2 + y_3^2}} \cos(\phi_2) = \frac{x_3^2 + y_3^2 + R_2^2 - R_1^2}{2R_2\sqrt{x_3^2 + y_3^2}} \tag{11.61}$$

$$a = \sin(\psi)\sin(\phi_2) + \cos(\psi)\cos(\phi_2)$$

Let,

$$a = \frac{x_3^2 + y_3^2 + R_2^2 - R_1^2}{2R_2\sqrt{x_3^2 + y_3^2}} \tag{11.62}$$

Becomes,

$$\cos(\phi_2 - \psi) = a \tag{11.63}$$

$$\phi_2 = \psi \pm \cos^{-1}(a) \tag{11.64}$$

$$= \operatorname{atan2}(y_3, x_3) \pm \operatorname{acos}\left(\frac{x_3^2 + y_3^2 + R_2^2 - R_1^2}{2R_2\sqrt{x_3^2 + y_3^2}}\right) \tag{11.65}$$

ϕ_1 can be obtained using equation (11.59)

$$\phi_1 = \operatorname{atan2}(\sin\phi_1, \cos\phi_1) \tag{11.66}$$

or use identity equation,

$$\tan(\frac{\phi}{2}) = \frac{1-\cos(\phi)}{\sin(\phi)} \text{ or } \tan(\frac{\phi}{2}) = \frac{\sin(\phi)}{1+\cos(\phi)} \tag{11.67}$$

$$\phi = 2\tan^{-1}(\frac{1-\cos(\phi)}{\sin(\phi)}) \text{ or } \phi = 2\tan^{-1}(\frac{\sin(\phi)}{1+\cos(\phi)} \tag{11.68}$$

Note
The function call

```
complexsolveRR(r1, r2, z3, phi1, phi2, phi1_2, phi2_2);
```

is equivalent to

```
complexsolve(2, 4, r1, r2, z3, phi1, phi2, phi1_2, phi2_2);
```

Example
For a complex equation

$$R_1 e^{i\phi_1} + R_2 e^{i\phi_2} = 1 + i2$$

Given $R_1 = 3$ and $R_2 = 4$, find ϕ_1 and ϕ_2

See Also
complexsolve(), **complexsolvePP**(), **complexsolvePR**(), **complexsolveRP**(), **complexsolveRRz**(), **polar**().

References

complexsolveRRz

Synopsis
#include <numeric.h>
int complexsolveRRz(double complex *z1*, **double complex** *r2*, **double complex** *z3*,
double &*phi1*, **double** &*phi2*, **double** &*phi1_2*, **double** &*phi2_2*);

Purpose
Solve for ϕ_1 and ϕ_2 in a complex equation in the polar format.

Return Value
This function returns the number of solutions with the value 2.

Parameters
z1 complex number on the left hand side.

z2 complex number on the left hand side.

z3 Complex number on the right hand side.

phi1 ϕ_1 for the first solution on the left hand side.

phi2 ϕ_2 for the first solution on the left hand side.

phi1_2 ϕ_1 for the second solution on the left hand side.

phi2_2 ϕ_2 for the second solution on the left hand side.

Description
This function is used to solve for ϕ_1 and ϕ_2 in a complex equation in polar format. The equation is in the form of

$$z_1 e^{i\phi_1} + z_2 e^{i\phi_2} = z_3 \tag{11.69}$$

Because it is a complex equation that can be partitioned into real and imaginary parts, two unknowns ϕ_1 and ϕ_2 can be solved in this equation.

Algorithm
Two unknowns can be solved by using equation (11.70) and its conjugate equation (11.71)

$$z_2 e^{i\phi_2} = z_3 - z_1 e^{i\phi_1} \tag{11.70}$$

$$\overline{z}_2 e^{-i\phi_2} = \overline{z}_3 - \overline{z}_1 e^{-i\phi_1} \tag{11.71}$$

Multiplying equations (11.70) and (11.71), we get

$$z_2\overline{z}_2 = z_3\overline{z}_3 - z_1\overline{z}_1 - z_3\overline{z}_1 e^{-i\phi_1} - \overline{z}_3 z_1 e^{i\phi_1} \tag{11.72}$$

Multiplying equation (11.72) by $e^{i\phi_1}$, we get

$$\overline{z}_3 z_1 (e^{i\phi_1})^2 + (z_2\overline{z}_2 - z_3\overline{z}_3 - z_1\overline{z}_1)e^{i\phi_1} + z_3\overline{z}_1 = 0 \tag{11.73}$$

We can write equation (11.73) as

$$ap^2 + bp + c = 0 \tag{11.74}$$

with $a = \overline{z}_3 z_1$, $b = z_2\overline{z}_2 - z_3\overline{z}_3 - z_1\overline{z}_1$, $c = z_3\overline{z}_1$, and $p = e^{i\phi_1}$. We can solve for two solutions of p using the complex quadratic equation (11.74). With

$$\cos\phi_1 = \mathrm{real}(p), \sin\phi_1 = \mathrm{imag}(p), \tag{11.75}$$

we can find $\phi_1 = \mathrm{atan2}(p_y, p_x)$.

For each ϕ_1, we can solve for ϕ_2 by equation (11.76)

$$e^{i\phi_2} = q = (z_3 - z_1 e^{i\phi_1})/z_2 \tag{11.76}$$

we can find $\phi_2 = \mathrm{atan2}(q_y, q_x)$.

Example

For a complex equation

$$z_1 e^{i\phi_1} + z_2 e^{i\phi_2} = z_3$$

Given $z_1 = 3$, $z_2 = 4$, and $z_3 = 1 + i2$, find ϕ_1 and ϕ_2

See Also

complexsolve(), **complexsolvePP**(), **complexsolvePR**(), **complexsolveRP**(), **complexsolveRR**(), **polar**().

References

condnum

Synopsis
#include <**numeric.h**>
double condnum(array double complex *a*[&][&]);

Purpose
Calculate the condition number of a matrix.

Return Value
This function returns the condition number.

Parameters
a Input matrix.

Description
The condition number of a matrix measures the sensitivity of the solution of a system of linear equations to errors in the data. It gives an indication of the accuracy of the results from matrix inversion and numerical solution of the linear system of equations solution. A value of the condition number near 1 indicates a well-conditioned matrix.

Algorithm
The function **condnum**() uses the the singular value decomposition function **svd**(). By function **svd**(), the matrix a is decomposed as

$$\mathbf{a} = \mathbf{usv^t}$$

The condition number is defined as $\max(s)/\min(s)$.

Example

```
#include <numeric.h>
int main() {
    array double a[2][2] = {2, 4,
                            3, 7};
    /* a2 is an ill-conditioned matrix  */
    array double a2[2][2] = {2, 4,
                             2.001, 4.0001};
    array float b[2][2] = {2, 4,
                           3, 7};

    array double complex z[2][2] = {2, complex(4, 3),
                                    3, 7};
    /* z2 is an ill-conditioned matrix */
    array double complex z2[2][2] = {2,     complex(4, 3),
                                     2.001, complex(4.0001,3)};
    array complex z3[2][2] = {2, complex(4, 3),
                              3, 7};
```

```
    double cond;

    cond = condnum(a);
    printf("condnum(a) = %g\n", cond);
    cond = condnum(a2);
    printf("condnum(a2) = %g\n", cond);

    cond = condnum(b);
    printf("condnum(b) = %g\n", cond);

    cond = condnum(z);
    printf("condnum(z) = %g\n", cond);
    cond = condnum(z2);
    printf("condnum(z2) = %g\n", cond);
    cond = condnum(z3);
    printf("condnum(z3) = %g\n", cond);
}
```

Output

```
condnum(a) = 38.9743
condnum(a2) = 10527.6
condnum(b) = 38.9743
condnum(z) = 9.32929
condnum(z2) = 11980.8
condnum(z3) = 9.32929
```

See Also

svd(), **norm**(), **rank**(), **rcondnum**().

References

conv

Synopsis
#include <numeric.h>
int conv(array double complex *c*[&], **array double complex** *x*[&], **array double complex** *y*[&]);

Syntax
conv(*c*, *x*, *y*)

Purpose
One-dimensional Discrete Fourier Transform (DFT) based convolution.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
c A one-dimensional array of size $n + m - 1$. It contains the result of convolution of *x* and *y*.

x A one-dimensional array of size n. It contains data used for convolution.

y A one-dimensional array of size m. It contains data used for convolution.

Description
The input **array** *x* and *y* can be of any supported arithmetic data type and sizes n and m, respectively. Conversion of the data to **double complex** type is performed internally. If both *x* and *y* are real type, the result is a one-dimensional array *c* of size $n + m - 1$. If either one of *x* and *y* is complex type, the result *c* is complex type.
If *x* and *y* are considered as two vectors of polynomial coefficients, the convolution of *x* and *y* is equivalent to the multiplication of these two polynomials.

Algorithm
The convolution of two functions $x(t)$ and $y(t)$, denoted as $x * y$, is defined by

$$x * y \equiv \int_{-\infty}^{\infty} x(\tau)y(t-\tau)d\tau$$

in the time domain with $x * y = y * x$. According to the theorem of convolution, if $X(f)$ and $Y(f)$ are Fourier transforms of $x(t)$ and $y(t)$, that is,

$$x(t) \Longleftrightarrow X(f) \qquad \text{and} \qquad y(t) \Longleftrightarrow Y(f)$$

then

$$x * y \Longleftrightarrow X(f)Y(f)$$

According to the theorem of discrete convolution,
if $X_d(f)$ and $Y_d(f)$ are discrete Fourier transforms of sequences $x[nT]$ and $y[nT]$, that is,

$$x[nT] \Longleftrightarrow X_d(f) \qquad \text{and} \qquad y[nT] \Longleftrightarrow Y_d(f)$$

then

$$x[nT] * y[nT] \Longleftrightarrow X_d(f)Y_d(f)$$

Where T is the sample interval and n is an index used in the summation.
Based on the theorem of discrete convolution, DFT can be used to compute discrete convolutions. The procedure is first to compute the DFT of $x[nT]$ and $y[nT]$ and then use the theorem to compute

$$c[nT] = x[nT] * y[nT] = F_d^{-1}[X_d(f)Y_d(f)]$$

to get the convolution result.
Suppose $x[nT]$ and $y[nT]$ have the same length N for n from 0 to $N-1$. Then the DFT of $x[nT]$ and $y[nT]$ have N points, while $c[nT]$ has length of $N+N-1$. Therefore, the signal produced by $x[nT] * y[nT]$ has length that is longer than the DFT length. The length difference produces time aliasing when the inverse DFT of $X_d(f)Y_d(f)$ is used to obtain $x[nT] * y[nT]$. Thus, $c[nt] = x[nT]y[nT]$ is a time-aliased version of the sequences convolution. To solve the time-aliasing problem, DFT can be used to compute discrete convolution of two signals if zero padded to the signals to extend their lengths to equal the discrete convolution length. That is, if the lengths of $x[nT]$ and $y[nT]$ are N_x and N_y, respectively, then $x[nT]$ and $y[nT]$ can be padded with N_y-1 and N_x-1 zeroes, respectively. This zero padding produces sample signals that are both of length N_x+N_y-1. DFT can be used with (N_x+N_y-1) points to compute $X_d(f)$ and $Y_d(f)$. The length, N_x+N_y-1, of the sample sequence obtained from the inverse DFT $F_d^{-1}[X_d(f)Y_d(f)]$ equals the length of the discrete convolution so that the discrete convolution is obtained.
In this numeric function, the sizes of two convolution arrays *x* and *y* are expanded to $m+n-1$ and zero padded internally. The DFT algorithm is used to compute the discrete Fourier transform of *x* and *y*. Multiplying two transforms together component by component, then using the inverse DFT algorithm to take the inverse discrete Fourier transform of the products, the answer is the convolution $x(nT) * y(nT)$.

Example 1
Assume $x(t)$ and $y(t)$ are polynomial functions, convolution of *x* and *y* is equivalent to multiplication of the two polynomials.

$$\begin{aligned} x(t) &= t^5 + 2*t^4 + 3*t^3 + 4*t^2 + 5*t + 6; \\ y(t) &= 6*t + 7; \end{aligned}$$

Convolution of $x(t) * y(t)$ or multiplication of polynomials $x(t)$ and $y(t)$ is equal to

$$c(t) = x(t) * y(t) = 6*t^6 + 19*t^5 + 32*t^4 + 45*t^3 + 58*t^2 + 71*t + 42$$

```
#include <stdio.h>
#include <numeric.h>
```

```
#define N 6        /* data x array size */
#define M 2        /* data y array size */
#define N2 (N+M-1)

int main() {
   int i;
   array double c[N2],x[N]={1,2,3,4,5,6},y[M]={6,7};

   conv(c,x,y); /* float data convolution */
   printf("Polynomial multiplication\n");
   printf("x=%6.3f\n",x);
   printf("y=%6.3f\n",y);
   printf("c=%6.3f\n",c);
}
```

Output

```
Polynomial multiplication
x= 1.000  2.000  3.000  4.000  5.000  6.000

y= 6.000  7.000

c= 6.000 19.000 32.000 45.000 58.000 71.000 42.000
```

Example 2

The sequences corresponding to the input and unit pulse response of a filter in a sample data control system are given as

$$x[n] = \begin{cases} n+1 & 0 \le n < 2 \\ 5-n & 2 \le n \le 3 \\ 0 & \text{elsewhere} \end{cases}$$

$$y[n] = \begin{cases} -n/2 & 2 \le n \le 4 \\ 0 & \text{elsewhere} \end{cases}$$

The convolution sum or output sequence becomes

$$\begin{aligned} c[n] = x[n] * y[n] &= \sum_{m=-\infty}^{\infty} x[m]y[n-m] \\ &= \begin{cases} \sum_{m=2}^{4} x[m]y[n-m] & 2 \le n \le 7 \\ 0 & \text{elsewhere} \end{cases} \end{aligned}$$

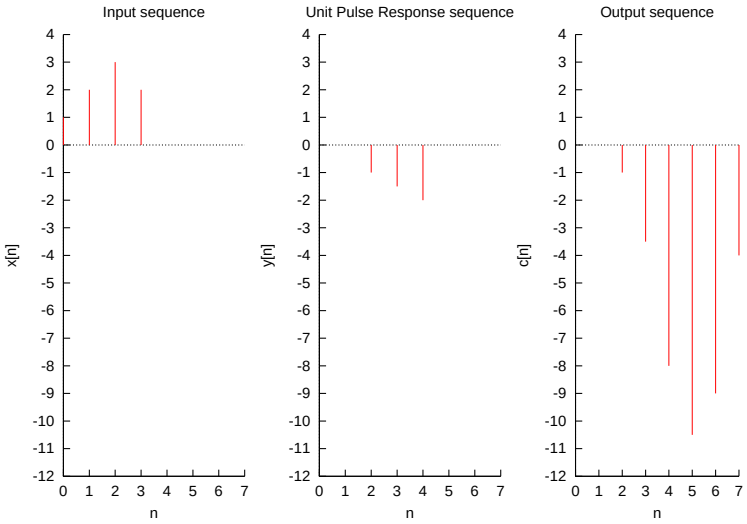
The six values of $c[n]$ for $i = 2$ to 7 can be calculated as follows

$$\begin{aligned} c[2] &= x[2]h[0] + x[3]h[-1] + x[4]h[-2] = -1 \\ c[3] &= x[2]h[1] + x[3]h[0] + x[4]h[-1] = -3.5 \\ c[4] &= x[2]h[2] + x[3]h[1] + x[4]h[0] = -8 \\ c[5] &= x[2]h[3] + x[3]h[2] + x[4]h[1] = -10.5 \\ c[6] &= x[2]h[4] + x[3]h[3] + x[4]h[2] = -9 \\ c[7] &= x[2]h[5] + x[3]h[4] + x[4]h[3] = -4 \end{aligned}$$

```
#include <stdio.h>
#include <chplot.h>
#include <numeric.h>

#define N 4        /* data x array size */
#define M 5        /* data y array size */
#define N2 (N+M-1)

int main() {
   int i,n1[N],n2[M],n3[N2];
   array double c[N2],x[N]={1,2,3,2},y[M]={0,0,-1,-1.5,-2};
   class CPlot plot;
   string_t labelx="x",
            labely="y";

   linspace(n1,0,N-1);
   linspace(n2,0,M-1);
   linspace(n3,0,N2-1);
   conv(c,x,y);
   printf("x=%6.3f\n",x);
   printf("y=%6.3f\n",y);
   printf("c=%6.3f\n",c);

   plot.data2D(n1,x);
   plot.plotType(PLOT_PLOTTYPE_IMPULSES, 0);
   plot.axisRange(PLOT_AXIS_X, 0,7,1);
   plot.axisRange(PLOT_AXIS_Y, -12,4,1);
   plot.label(PLOT_AXIS_X,"n");
   plot.label(PLOT_AXIS_Y,"x[n]");
   plot.title("Input sequence");
   plot.plotting();
   plot.deletePlots();
   plot.data2D(n2,y);
   plot.plotType(PLOT_PLOTTYPE_IMPULSES, 0);
   plot.axisRange(PLOT_AXIS_X, 0,7,1);
   plot.axisRange(PLOT_AXIS_Y, -12,4,1);
   plot.label(PLOT_AXIS_X,"n");
   plot.label(PLOT_AXIS_Y,"y[n]");
   plot.title("Unit Pulse Response sequence");
   plot.plotting();
   plot.deletePlots();
   plot.data2D(n3,c);
   plot.plotType(PLOT_PLOTTYPE_IMPULSES, 0);
   plot.axisRange(PLOT_AXIS_X, 0,7,1);
   plot.axisRange(PLOT_AXIS_Y, -12,4,1);
   plot.label(PLOT_AXIS_X,"n");
   plot.label(PLOT_AXIS_Y,"c[n]");
   plot.title("Output Response sequence");
   plot.plotting();
}
```

Output

```
x= 1.000  2.000  3.000  2.000

y= 0.000  0.000 -1.000 -1.500 -2.000

c= 0.000  0.000 -1.000 -3.500 -8.000 -10.500 -9.000 -4.000
```



See Also

fft(), **ifft**(), **conv2**(), **deconv**().

References

Nussbaumer, H. J, *Fast Fourier Transform and Convolution Algorithms*, New York: Springer-Verlay, 1982

conv2

Synopsis
#include <**numeric.h**>
int conv2(array double complex *c*[&][&], **array double complex** *f*[&][&],
array double complex *g*[&][&], ... /* [**string_t** *method*] */);

Syntax
conv2(*c*, *f*, *g*)
conv2(*c*, *g*, *g*, *method*);

Purpose
Two-dimensional Discrete Fourier Transform (DFT) based convolution.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
c A two-dimensional **array** with size of a section of $(na+nb-1)\times(ma+mb-1)$ specified by ”method”. It contains the result of convolution of *f* and *g*.

f A two-dimensional **array** of size $(na)\times(ma)$ used for convolution.

g A two-dimensional **array** of size $(nb)\times(mb)$ used for convolution.

method The **string_t** *method* used to specify the convolution result.
”full” - (default) returns the full 2-D convolution. The size is $(na+nb-1)\times(ma+mb-1)$.
”same” - returns the central part of 2-D convolution. That is the same size as *f*.
”valid” - returns only those parts of 2-D convolution that are computed without the zero-padded edges, the size of *c* is $(na-nb+1)\times(ma-mb+1)$ where the size of *f* must be bigger than the size of *g*. That is, size(*f*) $>$ size(*g*).

Description
The input two-dimensional **array** *f* and *g* can be of any supported arithmetic data type and sizes, respectively. Conversion of the data to **double complex** is performed internally. If both of *f* and *g* are **real** data, the result is a two-dimensional **real array** *c*. If either one of *f* and *g* is **complex** , the result **array** *c* will be **complex** data type.
If size of *f* and size of *g* are $(ma)\times(na)$ and $(mb)\times(nb)$, respectively, then
method = ”full” (default) the size of *c* is $(ma+mb-1)\times(na+nb-1)$;
method = ”same” the size of *c* is $(ma)\times(na)$;
method = ”valid” the size of *c* is $(ma-mb+1)\times(na-nb+1)$ where the size of *f* must be bigger than the

size of *g*. That is, size(*f*)> size(*g*).

Algorithm

Two-dimensional convolution is analogous in form to one-dimensional convolution. Thus for two functions $f(x, y)$ and $g(x, y)$, we define

$$f(x, y) * g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\alpha, \beta) g(x - \alpha, y - \beta) d\alpha d\beta$$

The convolution theorem in two dimensions, then, is given by the relations

$$f(x, y) * g(x, y) \Longleftrightarrow F(u, v)G(u, v)$$

where $F(u, v)$ and $G(u, v)$ are two-dimensional Fourier transform of $f(x, y)$ and $g(x, y)$, respectively. The discrete convolution of the two functions $f(mT, nT)$ and $g(mT, nT)$ is given by the relation

$$f(mT, nT) * g(mT, nT) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(mT, nT) g(k_1 - m, k_2 - n)$$

for $k_1 = 0, 1, 2, \cdots, M - 1$ and $k_2 = 0, 1, 2, \cdots, N - 1$. Where T is the sampling interval, M and N are periods in x and y directions, respectively.
Similarly, the discrete convolution theorem in two dimensions is given as

$$c(mT, nT) = f(mT, nT) * g(mT, nT) \Longleftrightarrow F_d(f_1, f_2)G_d(f_1, f_2)$$

where $F_d(f_1, f_2)$ and $G_d(f_1, f_2)$ are discrete Fourier transform of $f(mT, nT)$ and $g(mT, nT)$, respectively. We can compute the DFT of $f(mT, nT)$ and $g(mT, nT)$ and then use the convolution theorem to compute

$$c(mT, nT) = f(mT, nT) * g(mT, nT) = F_d^{-1}[F_d(f_1, f_2)G_d(f_1, f_2)]$$

to get the two-dimensional result.
Similar to the one-dimensional convolution, if periods M and N are used to compute the convolution, there is a time-aliasing in the signals. To solve the time-aliasing problem, zeroes are padded to the signal to extend their lengths to equal the discrete convolution length. That is, if the lengths of $f(nT, mT)$ is $(na) \times (ma)$ and $g(nT, mT)$ is $(nb) \times (mb)$, then $f(nT, mT)$ and $g(nT, mT)$ are padded with $(nb - 1) \times (mb - 1)$ and $(na - 1) \times (ma - 1)$ zeroes, respectively. This zero padding produces sample signals that are both of length $(na + nb - 1) \times (ma + mb - 1)$. A two-dimensional DFT with $(na + nb - 1) \times (ma + mb - 1)$ points is used to compute $F_d(f_1, f_2)$ and $G_D(F_1, F_2)$. The length, $(na + nb - 1) \times (ma + mb - 1)$, of the sample sequence obtained from the inverse DFT $F_d^{-1}[F_d(f_1, f_2)G_d(f_1, f_2)]$ equals the length of the discrete convolution so that the discrete convolution is produced.
In this numeric function, the size of two convolution arrays *f* and *g* are expanded to $(na + nb - 1) \times (ma + mb - 1)$ and zero padded internally. The DFT algorithm is used to compute the two-dimensional discrete Fourier transform of *f* and *g*. Multiplying two transforms together component by component, then using the inverse DFT algorithm to take the inverse discrete Fourier transform of the products, the answer is the two-dimensional convolution $f(mT, nT) * g(mT, nT)$.

Example

In image processing, a Sobel filter is a simple approximation to the concept of edge detection. Convoluting the original two-dimensional image data with a Sobel filter

$$g_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

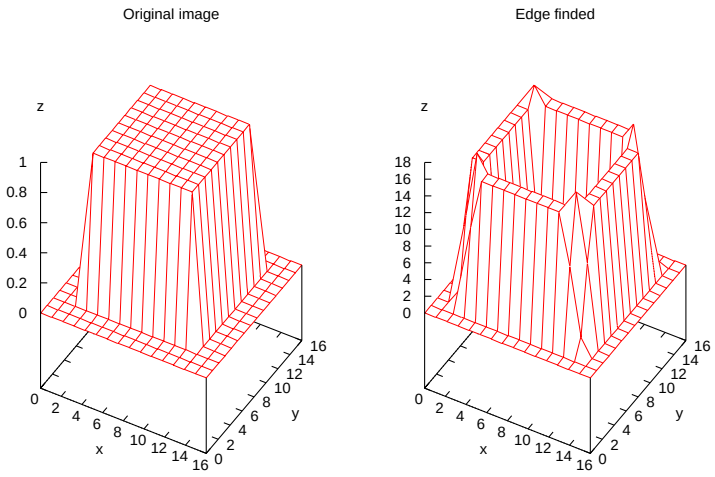
detects the edges in x-direction. Similarly, convoluting a Sobel filter

$$g_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

which is the transpose of matrix g_x, edges in y-direction can be detected.

```
#include <math.h>
#include <chplot.h>
#include <numeric.h>

int main() {
   int i, j;
   array double g[3][3]={{-1,0,1},{-2,0,2},{-1,0,1}};
   array double x[16], y[16], z1[256], f[16][16], H[18][18], V[18][18],Z[18][18], Z1[256];

   linspace(x,0,16);
   linspace(y,0,16);
   for(i=3; i<13; i++)
      for(j=3; j<13; j++) {
         z1[i*16+j]=1;
         f[i][j] = 1;
   }
   plotxyz(x,y,z1);                    /* original image */
   conv2(H,f,g);
   conv2(V,f,transpose(g));
   Z = H .* H + V .* V;                /* magnitude of the pixel value */
   for(i = 0; i<16; i++)
      for(j=0; j<16; j++)
         Z1[i*16+j] = Z[i+1][j+1];
   plotxyz(x,y,Z1);                    /* edge finded image */
}
```

Output



See Also

fft(), **ifft**(), **conv**(), **deconv**().

References

Rafael C. Gonzalez, Paul Wintz, Digital Image Processing, Second Edition, Addison-Wesley Publishing Company, 1987.

corrcoef

Synopsis
#include <numeric.h>
int corrcoef(array double &*c*, **array double** &*x*, ... /* [**array double** &*y*] */);

Syntax
corrcoef(*c*, *x*)
corrcoef(*c*, *x*, *y*)

Purpose
Correlation coefficients calculation.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
c Correlation coefficient matrix formed from array *x*.

x A vector of observation or a two-dimensional **array** whose column is an observation and row is a variable.

y Optional argument. *y* is a vector or two-dimensional array whose length or column length shall be the same as the column length of *x*.

Description
The input array *x* can be of any supported arithmetic data type of vector or two-dimensional array of any size $n \times m$ (if *x* is a vector, regard the size as $1 \times m$). Each column of *x* is an observation and each row is a variable. Optional input *y* can only be of **double** data type and same column length as *x*. That is, the size of *y* is $n_1 \times m$ or regard as $1 \times m$ if *y* is a vector. Attachment of *y* to row a of *x* as a new expanding matrix [**X**] is performed internally. **corrcoef**(*c*, *x*, *y*) is equivalent to **corrcoef**(*c*,[**X**]) where **X** is defined as $\mathbf{X} = \begin{pmatrix} x \\ y \end{pmatrix}$. The size of $[\mathbf{X}]$ is $(n + n_1) \times m$. Conversion of the data to **double** is performed internally. The result *c* is a matrix of correlation coefficients.

Algorithm
For array *x* of size $N \times M$, which has N variable and each variable has total M observation, correlation coefficient matrix *c* is related to the covariance matrix C by

$$c_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} * C_{jj}}}$$

where C is a covariance matrix. It is calculated by

$$C = \frac{1}{N-1} u * u'$$

where element u is defined as

$$u_{ij} = x_{ij} - \mu_i; \qquad i = 0, 1, \cdots, N; j = 0, 1, \cdots, M$$

and

$$\mu_i = \frac{1}{M} \sum_{j=1}^{M} x_{ij}; \qquad i = 0, 1, \cdots, N$$

where x_{ij} is the element of the input matrix.

Example

Consider an array $x[4][5]$.

$$x = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 3 & 4 & 5 & 6 \end{bmatrix}$$

where each row of the x is a variable and column is an observation. The result array c is the correlation coefficient matrix.

```
#include <stdio.h>
#include <numeric.h>

#define N 4         /* data array size */
#define N1 5
#define N2 4

int main(){
   int i,j;
   array double  c[N2][N2];
   array double  x[N][N1]={{1,2,3,4,5},
                           {1,2,3,4,5},
                           {0,0,0,0,1},
                           {2,3,4,5,6}};
   corrcoef(c,x);
   printf("x=%6.3f\n",x);
   printf("correlation coefficient matrix\n");
   printf("%6.3f\n",c);
}
```

Output

```
x= 1.000  2.000  3.000  4.000  5.000
 1.000  2.000  3.000  4.000  5.000
 0.000  0.000  0.000  0.000  1.000
 2.000  3.000  4.000  5.000  6.000
```

```
correlation coefficient matrix
 1.000  1.000  0.707  1.000
 1.000  1.000  0.707  1.000
 0.707  0.707  1.000  0.707
 1.000  1.000  0.707  1.000
```

See Also
covariance().

correlation2

Synopsis
#include <**numeric.h**>
double correlation2(**array double** *x*[&][&], **array double** *y*[&][&]);

Syntax
correlation2(*x*, *y*)

Purpose
Two-dimensional correlation coefficient.

Return Value
This function returns the two-dimensional correlation coefficient.

Parameters
x A square matrix of size $n \times n$. It contains the original date for correlation coefficient calculation.

y A square matrix of the same size as *x*. It contains the original date for correlation coefficient calculation.

Algorithm
This function is used to calculate the two-dimensional correlation coefficient. The two-dimensional correlation coefficient is defined as

$$c = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(xx_{ij} * yy_{ij})}{\sqrt{(\sum_{i=1}^{n}\sum_{j=1}^{n} xx_{ij}^2) * (\sum_{i=1}^{n}\sum_{j=1}^{n} yy_{ij}^2)}}$$

where

$$xx_{ij} = x_{ij} - \mu_x$$
$$yy_{ij} = y_{ij} - \mu_y$$

μ_x and μ_y are the mean values of the matrix *x* and *y*.

$$\mu_x = \frac{1}{n * n}\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}$$
$$\mu_y = \frac{1}{n * n}\sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij}$$

Example
Calculate two square matrix correlation coefficient.

```
#include <stdio.h>
#include <numeric.h>

#define NC 6
#define ND 3
#define N 20

int main() {
   int i,j,k;
   array double x[3][3]={1,2,3,
                         3,4,5,
                         6,7,8};
   array double y[3][3]={3,2,2,
                         3,8,5,
                         6,2,5};
   double c;
   c = correlation2(x,y);
   printf("x=\n%f",x);
   printf("y=\n%f",y);
   printf("c=\n%f\n",c);
}
```

Output

```
x=
1.000000 2.000000 3.000000
3.000000 4.000000 5.000000
6.000000 7.000000 8.000000
y=
3.000000 2.000000 2.000000
3.000000 8.000000 5.000000
6.000000 2.000000 5.000000
c=
0.326637
```

See Also

correlation().

covariance

Synopsis
#include <**numeric.h**>
int covariance(**array double** &*c*, **array double** &*x*, ... /* [**array double** &*y*] */);

Syntax
covariance(*c*, *x*)
covariance(*c*, *x*, *y*)

Purpose
Covariance matrix.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

c Covariance matrix. If there is only one input *x* and it is a vector, this is the variance. If *x* is a matrix, where each column is an observation and each row is a variable, *c* is the covariance matrix. The diagonal of *c* is a vector of variances for each row. The square root of diagonal of *c* is a vector of standard deviations.

x A vector whose elements are an observation, or a two-dimensional **array** whose column is an observation and row is a variable.

y Optional argument. *y* could be a vector or a two-dimensional **array** whose length or column length shall be same as the column length of *x*.

Description
The input **array** *x* can be of any supported arithmetic data type of vector or matrix of any size $n \times m$ (if *x* is a vector, regard the size as $1 \times m$). Each column of *x* is an observation and each row is a variable. Optional input *y* can only be of **double** data type and same column length as *x*. That is, the size of *y* is $n_1 \times m$ or regard as $1 \times m$ if *y* is a vector. Attachment of *y* to row of *x* as a new expanding matrix [**X**] is performed internally. **covariance**(*c*, *x*, *y*) is equivalent to **covariance**(*c*,[**X**]) where **X** is defined as $\mathbf{X} = \begin{pmatrix} x \\ y \end{pmatrix}$. The size of $[\mathbf{X}]$ is $(n + n_1) \times (m)$. Conversion of the data to **double** is performed internally. The result *c* is a variance or a covariance matrix depending on *x* and *y*. If *x* is a vector and no optional input *y*, the result *c* is a variance. Otherwise it is a covariance. Diagonal of *c* is a vector of variances for each row. The square root of the diagonal of *c* is a vector of standard deviations.

Algorithm

For each of **array** *x*$[N][M]$ of N variable which has total M observation, covariance function is defined as

$$covariance(x_i, x_j) = E[(x_i - \mu_i)(x_j - \mu_j)] \qquad i, j = 1, 2, \cdots, N;$$

where x_i and x_j are the *ith* and *jth* row elements. E is the mathematical expectation and $\mu_i = Ex_i$.
In function **covariance**(), the mean from each column is removed before calculating the result.

$$u_{ij} = x_{ij} - \mu_i; \qquad i = 0, 1, \cdots, N; j = 0, 1, \cdots, M$$

where

$$\mu_i = \frac{1}{M} \sum_{j=1}^{M} x_{ij}; \qquad i = 0, 1, \cdots, N$$

Then the covariance matrix is calculated by

$$c = \frac{1}{N-1} u * u'$$

Example

Consider an array $x[4][5]$.

$$x = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 3 & 4 & 5 & 6 \end{bmatrix}$$

where each row of the x is a variable and column is an observation. The result **array** c is the covariance matrix. The diagonal elements $c(i, j)$ represent the variances for the rows of x. The off-diagonal element $c(i, j)$ represents the covariances of rows i and j.

```
#include <stdio.h>
#include <numeric.h>

#define N 4         /* data array size */
#define N1 5
#define N2 4

int main() {
   int i,j;
   array double  c[N2][N2];
   array double  x[N][N1]={{1,2,3,4,5},
                           {1,2,3,4,5},
                           {0,0,0,0,1},
                           {2,3,4,5,6}};

   covariance(c,x);     /* covariance of matrix */

   printf("x=%8.3f\n",x);
   printf("c=%8.3f\n",c);
}
```

Output

```
x=   1.000    2.000    3.000    4.000    5.000
   1.000    2.000    3.000    4.000    5.000
   0.000    0.000    0.000    0.000    1.000
   2.000    3.000    4.000    5.000    6.000

c=   2.500    2.500    0.500    2.500
   2.500    2.500    0.500    2.500
   0.500    0.500    0.200    0.500
   2.500    2.500    0.500    2.500
```

See Also
corrcoef().

cpolyeval

Synopsis
#include <numeric.h>
double complex cpolyeval(array double complex *c*[&], **double complex** *z*);

Syntax
cpolyeval(*c*, *z*)

Purpose
Calculate the value of a polynomial at point *z*.

Return Value
This function returns the **complex** value of a polynomial at a given complex point *z*.

Parameters
c A vector of coefficients of a polynomial.

z A value of point x in which polynomial is evaluated.

Description
The vector of coefficient of polynomial *c* can be of any supported arithmetic data type and extent size. The value of *z* can be any data type. Conversion of the data to **double complex** is performed internally. The return value is **double complex** data of the value of the polynomial.

Algorithm
For polynomial

$$P(z) = c_0 z^n + c_1 z^{n-1} + c_2 z^{n-1} + \cdots + c_{n-1} z + c_n$$

The calculation of the polynomial value is implemented using the follow algorithm

$$P = (((\cdots(c_0 z + c_1)z + c_2)z + c_3)z + \cdots + c_{n-1})z + c_n$$

Example
Evaluate polynormal

$$P(z) = z^5 - 5z^4 + 10z^3 - 10z^2 + 5z - 1$$

at complex point $z = 0.1 + i0.1$.

```
#include <stdio.h>
#include <numeric.h>
```

```
#define NC 6
#define ND 3

int main() {
   int i,j,k;
   double c[NC]={1.0,-5.0,10.0,-10.0,5.0,-1.0};
   double complex z=complex(0.1,1),val;

   val = cpolyeval(c,z);
   printf("z = %f   function value = %f\n",z,val);
}
```

Output

```
z = complex(0.100000,1.000000)   function value = complex(2.199510,-3.819500)
```

see also

polyeval(), **polyevalarray**().

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press, 1997.

cproduct

Synopsis
#include <**numeric.h**>
double complex cproduct(**array double complex** &*a*, ... /* [**array double complex** *v*[:]] */);

Syntax
cproduct(a)
cproduct(a, v)

Purpose
Calculate products of all elements and products of the elements of each rows of a complex array.

Return Value
This function returns the product of all elements.

Parameters
a The input complex array for which the products are calculated.

v The output array which contains the products of each row of a two-dimensional complex array.

Description
This function calculates the product of all elements in a complex array of any dimension. If the array is a two-dimensional matrix, the function can calculate the products of each row. The product of all elements of the array is returned by the function and the products of the elements of each row are passed out by the argument *v*. The input array can be of any dimension and any arithmetic data types.

Example1
Calculate the products of all elements of the complex arrays with different data types and dimensions.

```
#include <numeric.h>
int main() {
    complex double a[3] = {complex(1,2), complex(2,3), complex(1,2)};
    complex double b[2][3] = {1,2,3,4,5,6};
    int b1[2][3] = {1,2,3,4,5,6};
    double c[2][3][5];
    c[0][0][0] = 10;
    c[0][0][1] = 20;
    double complex products;

    products = cproduct(a);
    printf("cproduct(a) = %f\n", products);
    products = cproduct(b);
    printf("cproduct(b) = %f\n", products);
    products = cproduct(b1);
    printf("cproduct(b) = %f\n", products);
    products = cproduct(c);
    printf("cproduct(c) = %f\n", products);
}
```

Output1

```
cproduct(a) = complex(-18.000000,-1.000000)
cproduct(b) = complex(720.000000,0.000000)
cproduct(b) = complex(720.000000,0.000000)
cproduct(c) = complex(0.000000,0.000000)
```

Example2

Calculate the products of the elements of each row of two-dimensional complex arrays with different data types and dimensions.

```
#include <numeric.h>
int main() {
    double complex a[2][3] = {complex(1,2),2,3,
                      4,5,6};
    array double complex b[3][4] = {1,2,3,4,
                            5,6,7,8,
                            1,2,3,4};
    array double complex cproductv1[2], cproductv2[3];

    cproduct(a,cproductv1);
    printf("cproduct(a, cproductv1) = %f\n", cproductv1);
    cproduct(b,cproductv2);
    printf("cproduct(b,cproductv2) = %f\n", cproductv2);
}
```

Output2

```
cproduct(a, cproductv1) = complex(6.000000,12.000000) complex(120.000000,0.000000)

cproduct(b,cproductv2) = complex(24.000000,0.000000) complex(1680.000000,0.000000) complex(24.000
```

See Also

product(), **cumprod**(), **mean**(), **median**(), **sum**(), **cumsum**().

References

cross

Synopsis

#include <**numeric.h**>
array double cross(**array double** *a*[&], **array double** *b*[&])[3];

Purpose

Calculate the cross product of two vectors.

Return Value

This function returns the cross product of vectors *a* and *b*.

Parameters

a An input array which contains the first vector.

b An input array which contains the second vector.

Description

The function **cross()** returns the cross product of vectors *a* and *b*. The number of elements of each vector shall be 3.

Example

```
#include <numeric.h>
int main() {
    array double a[3] = {1, 2, 3};
    array double b[ ] = {1, 2, 3};
    array double c[3] = {2, 3, 4};
    array double crossprod[3];

    crossprod = cross(a, b);
    printf("cross(a,b) = %f\n", crossprod);
    crossprod = cross(a, c);
    printf("cross(a,c) = %f\n", crossprod);
}
```

Output

```
cross(a,b) = 0.000000 0.000000 0.000000

cross(a,c) = -1.000000 2.000000 -1.000000
```

See Also

dot().

References

csum

Synopsis
#include <**numeric.h**>
double complex csum(**array double complex** &*a*, ... /*[**array double complex** *v*[:]]*/);

Syntax
csum(a)
csum(a, v)

Purpose
Calculate the sum of all elements of an array and the sums of each row of a two-dimensional complex array.

Return Value
This function returns the sum of all elements of the complex array.

Parameters
a The input complex array.

v The output array which contains the sums of each row of array.

Description
The function calculates the sum of all elements in a complex array of any dimension. If the array is a two-dimensional matrix, the function can calculate the sum of each row. The input array can be any arithmetic data type and of any dimension.

Example1
Calculate the sum of all elements of a complex arrays of different data types.

```
#include <numeric.h>
int main() {
    complex double a[3] = {complex(1,2), complex(2,3)};
    complex double b[2][3] = {1,2,3,4,5,6};
    int b1[2][3] = {1,2,3,4,5,6};
    double c[2][3][5];
    c[0][0][0] = 10;
    c[0][0][1] = 20;
    double complex sums;

    sums = csum(a);
    printf("csum(a) = %f\n", sums);
    sums = csum(b);
    printf("csum(b) = %f\n", sums);
    sums = csum(b1);
    printf("csum(b) = %f\n", sums);
    sums = csum(c);
    printf("csum(c) = %f\n", sums);
}
```

Output1

```
csum(a) = complex(3.000000,5.000000)
csum(b) = complex(21.000000,0.000000)
csum(b) = complex(21.000000,0.000000)
csum(c) = complex(30.000000,0.000000)
```

Example2
Calculate the sums of each elements of the complex arrays

```
#include <numeric.h>
int main() {
    double complex a[2][3] = {complex(1,2),2,3,
                        4,5,6};
    array double complex b[3][4] = {1,2,3,4,
                             5,6,7,8,
                             1,2,3,4};
    array int    b1[3][4] = {1,2,3,4,
                             5,6,7,8,
                             1,2,3,4};
    array double complex csumv1[2], csumv2[3];

    csum(a, csumv1);
    printf("csum(a, csumv1) = %.3f\n", csumv1);
    csum(b, csumv2);
    printf("csum(b, csumv2) = %.3f\n", csumv2);
    csum(b1, csumv2);
    printf("csum(b1, csumv2) = %.3f\n", csumv2);
}
```

Output2

```
csum(a, csumv1) = complex(6.000,2.000) complex(15.000,0.000)

csum(b, csumv2) = complex(10.000,0.000) complex(26.000,0.000) complex(10.000,0.000)

csum(b1, csumv2) = complex(10.000,0.000) complex(26.000,0.000) complex(10.000,0.000)
```

See Also

sum(), **cumsum**(), **trace**(), **product**().

References

ctrace

Synopsis

#include <**numeric.h**>
double complex ctrace(array double complex *a*[&][&]);

Purpose

Calculate the sum of diagonal elements of a complex matrix.

Return Value

This function returns the sum.

Parameters

a Input two-dimensional complex array.

Description

This function calculates the sum of the diagonal elements of the complex matrix.

Algorithm

For a matrix *a*, the trace is defined as

$$trace = \sum_{i=1}^{n} a_{ii}$$

Example

```
#include <numeric.h>
int main() {
    array double complex a[2][2] = {complex(2,3), -4,
                                    3, -7};
    array double complex a2[3][2] = {complex(2,3), -4,
                                    3, -7,
                                    1, 1};
    array double b[2][2] = {2, -4,
                            3, -7};
    double complex t;

    t = ctrace(a);
    printf("ctrace(a) = %f\n", t);
    t = ctrace(a2);
    printf("ctrace(a2) = %f\n", t);
    t = ctrace(b);
    printf("ctrace(b) = %f\n", t);
}
```

Output

```
ctrace(a) = complex(-5.000000,3.000000)
ctrace(a2) = complex(-5.000000,3.000000)
ctrace(b) = complex(-5.000000,0.000000)
```

See Also
trace(), **sum**(), **mean**(), **median**().

References

ctriangularmatrix

Synopsis
#include <**numeric.h**>
array complex double ctriangularmatrix(string_t *pos*, **array double complex** *a*[&][&], ... /* [**int** k] */)[:][:];

Syntax
ctriangularmatrix(*pos*, *a*)
ctriangularmatrix(*pos*, *a*, *k*)

Purpose
Get the upper or lower triangular part of a complex matrix.

Return Value
This function returns the upper or lower triangular part of a complex matrix.

Parameters
pos Input string indicating which part of the matrix to be returned.

a Input 2-dimensional array.

k Input integer. A matrix is returned on and below the *k*th diagonal of the input matrix.

Description
This function gets the triangular matrix on and below the *k*th diagonal of matrix *a*. For *pos* of “upper” the function returns the upper complex triangular part of the matrix, and for *pos* of “lower” the function returns the lower complex part of the matrix. *k* indicates the offset of the triangular matrix to the lower *k*th diagonal of the matrix.

Example

```
#include <numeric.h>
int main() {
    array double complex a[4][3] = {1,2,3,
                                    4,5,6,
                                    7,8,9,
                                    3,6,4};
    int n = 4, m=3, k;
    array double complex t[n][m];


    t = ctriangularmatrix("upper",a);
    printf("ctriangularmatrix(\"upper\", t) =\n%5.3f\n", t);

    k = 0;
    t = ctriangularmatrix("upper",a,k);
    printf("ctriangularmatrix(\"upper\", t, 0) =\n%5.3f\n", t);
```

```
    k = 1;
    t = ctriangularmatrix("upper",a,k);
    printf("ctriangularmatrix(\"upper\", t, 1) =\n%5.3f\n", t);

    k = -1;
    t = ctriangularmatrix("upper",a,k);
    printf("ctriangularmatrix(\"upper\", t, -1) =\n%5.3f\n", t);


    t = ctriangularmatrix("lower",a);
    printf("ctriangularmatrix(\"lower\", t) =\n%5.3f\n", t);

    k = 0;
    t = ctriangularmatrix("lower",a,k);
    printf("ctriangularmatrix(\"lower\", t, 0) =\n%5.3f\n", t);

    k = 1;
    t = ctriangularmatrix("lower",a,k);
    printf("ctriangularmatrix(\"lower\", t, 1) =\n%5.3f\n", t);

    k = -1;
    t = ctriangularmatrix("lower",a,k);
    printf("ctriangularmatrix(\"lower\", t, -1) =\n%5.3f\n", t);
}
```

Output

```
ctriangularmatrix("upper", t) =
complex(1.000,0.000) complex(2.000,0.000) complex(3.000,0.000)
complex(0.000,0.000) complex(5.000,0.000) complex(6.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(9.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(0.000,0.000)

ctriangularmatrix("upper", t, 0) =
complex(1.000,0.000) complex(2.000,0.000) complex(3.000,0.000)
complex(0.000,0.000) complex(5.000,0.000) complex(6.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(9.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(0.000,0.000)

ctriangularmatrix("upper", t, 1) =
complex(0.000,0.000) complex(2.000,0.000) complex(3.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(6.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(0.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(0.000,0.000)

ctriangularmatrix("upper", t, -1) =
complex(1.000,0.000) complex(2.000,0.000) complex(3.000,0.000)
complex(4.000,0.000) complex(5.000,0.000) complex(6.000,0.000)
complex(0.000,0.000) complex(8.000,0.000) complex(9.000,0.000)
complex(0.000,0.000) complex(0.000,0.000) complex(4.000,0.000)

ctriangularmatrix("lower", t) =
complex(1.000,0.000) complex(0.000,0.000) complex(0.000,0.000)
complex(4.000,0.000) complex(5.000,0.000) complex(0.000,0.000)
complex(7.000,0.000) complex(8.000,0.000) complex(9.000,0.000)
complex(3.000,0.000) complex(6.000,0.000) complex(4.000,0.000)

ctriangularmatrix("lower", t, 0) =
complex(1.000,0.000) complex(0.000,0.000) complex(0.000,0.000)
```

```
complex(4.000,0.000) complex(5.000,0.000) complex(0.000,0.000)
complex(7.000,0.000) complex(8.000,0.000) complex(9.000,0.000)
complex(3.000,0.000) complex(6.000,0.000) complex(4.000,0.000)

ctriangularmatrix("lower", t, 1) =
complex(1.000,0.000) complex(2.000,0.000) complex(0.000,0.000)
complex(4.000,0.000) complex(5.000,0.000) complex(6.000,0.000)
complex(7.000,0.000) complex(8.000,0.000) complex(9.000,0.000)
complex(3.000,0.000) complex(6.000,0.000) complex(4.000,0.000)

ctriangularmatrix("lower", t, -1) =
complex(0.000,0.000) complex(0.000,0.000) complex(0.000,0.000)
complex(4.000,0.000) complex(0.000,0.000) complex(0.000,0.000)
complex(7.000,0.000) complex(8.000,0.000) complex(0.000,0.000)
complex(3.000,0.000) complex(6.000,0.000) complex(4.000,0.000)
```

See Also

triangularmatrix(), **diagonal**(), **cdiagonal**()

cumprod

Synopsis
#include <**numeric.h**>
int cumprod(**array double complex** &*y*, **array double complex** &*x*);

Syntax
cumprod(*y*, *x*)

Purpose
Calculate cumulative product of elements in an array.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Input array with dimension less than three. It contains data to be calculated.

y Output array of same dimension as *x*. It contains the result of cumulative product.

Description
This function computes the cumulative product of the input array *x*. If the input *x* is a vector, it calculates the cumulative product of the elements of *x*. If the input *x* is a two-dimensional matrix, it calculates the cumulative product over each row. If the input *x* is a three-dimensional array, it calculates the cumulative product over the the first dimension. It is invalid for calculation of cumulative product for arrays with more than three dimensions.

Algorithm
For a vector $x = [x_1, x_2, \cdots, x_n]$, cumulative productive vector $y = [y_1, y_2, \cdots, y_n]$ is defined by

$$y_i = x_1 * x_2 * \cdots * x_i \qquad i = 1, 2, \cdots, n$$

Example
Calculation of cumulative products of arrays of different dimensions.

```
#include <numeric.h>
int main() {
   array double x[6]={1,2,3,4,5,6}, y[6];
   array double complex zx[2][3]={complex(1,1),2,3,complex(2,2),5,6}, zy[2][3];
   array double x1[2][3][4]={{1,2,3,4,
                              5,6,7,8,
                              5,6,7,8},
                              {10,11,12,13,
                               10,11,12,13,
                               14,15,16,17}};
   array double y1[2][3][4];
```

```
   cumprod(y,x);
   printf("x=%f",x);
   printf("y=%f",y);
   printf("\n");

   cumprod(zy,zx);
   printf("zx=%5.2f",zx);
   printf("zy=%5.2f",zy);
   printf("\n");

   cumprod(y1,x1);
   printf("x1=%f",x1);
   printf("y1=%f",y1);
}
```

Output

```
x=1.000000 2.000000 3.000000 4.000000 5.000000 6.000000
y=1.000000 2.000000 6.000000 24.000000 120.000000 720.000000

zx=complex( 1.00, 1.00) complex( 2.00, 0.00) complex( 3.00, 0.00)
complex( 2.00, 2.00) complex( 5.00, 0.00) complex( 6.00, 0.00)
zy=complex( 1.00, 1.00) complex( 2.00, 2.00) complex( 6.00, 6.00)
complex( 2.00, 2.00) complex(10.00,10.00) complex(60.00,60.00)

x1=1.000000 2.000000 3.000000 4.000000
5.000000 6.000000 7.000000 8.000000
5.000000 6.000000 7.000000 8.000000

10.000000 11.000000 12.000000 13.000000
10.000000 11.000000 12.000000 13.000000
14.000000 15.000000 16.000000 17.000000
y1=1.000000 2.000000 6.000000 24.000000
5.000000 30.000000 210.000000 1680.000000
5.000000 30.000000 210.000000 1680.000000

10.000000 110.000000 1320.000000 17160.000000
10.000000 110.000000 1320.000000 17160.000000
14.000000 210.000000 3360.000000 57120.000000
```

See Also

cumsum(), **product**(), **cproduct**(), **sum**(), **csum**().

cumsum

Synopsis
#include <**numeric.h**>
int cumsum(**array double complex** *&y*, **array double complex** *&x*);

Syntax
cumsum(*y*, *x*)

Purpose
Calculate cumulative sum of elements in an array.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Input array whose dimension are no more than three. It contains data to be calculated.

y Output array of same dimensional as *x*. It contains data the result of cumulative sum.

Description
This function computes the cumulative sum of the input array *x*. If the input *x* is a vector, it calculates the cumulative sum of the elements of *x*. If the input *x* is a two-dimensional matrix, it calculates the cumulative sum over each row. If the input *x* is a three-dimensional array, it calculates the cumulative sum over the first dimension. It is invalid for calculation of cumulative sum for arrays with more than three dimensions.

Algorithm
For a vector $x = [x_1, x_2, \cdots, x_n]$, cumulative sum vector $y = [y_1, y_2, \cdots, y_n]$ is defined by

$$y_i = x_1 + x_2 + \cdots + x_i \qquad i = 1, 2, \cdots, n$$

Example
Calculation of cumulative sum of arrays of different dimensions.

```
#include <numeric.h>
int main() {
   array double x[6]={1,2,3,4,5,6}, y[6];
   array double complex zx[2][3]={complex(1,1),2,3,complex(2,2),5,6}, zy[2][3];
   array double x1[2][3][4]={{1,2,3,4,
                              5,6,7,8,
                              5,6,7,8},
                              {10,11,12,13,
                               10,11,12,13,
                               14,15,16,17}};
   array double y1[2][3][4];
```

```
   cumsum(y,x);
   printf("x=%f",x);
   printf("y=%f",y);
   printf("\n");

   cumsum(zy,zx);
   printf("zx=%5.2f",zx);
   printf("zy=%5.2f",zy);
   printf("\n");

   cumsum(y1,x1);
   printf("x1=%f",x1);
   printf("y1=%f",y1);
}
```

Output

```
x=1.000000 2.000000 3.000000 4.000000 5.000000 6.000000
y=1.000000 3.000000 6.000000 10.000000 15.000000 21.000000

zx=complex( 1.00, 1.00) complex( 2.00, 0.00) complex( 3.00, 0.00)
complex( 2.00, 2.00) complex( 5.00, 0.00) complex( 6.00, 0.00)
zy=complex( 1.00, 1.00) complex( 3.00, 1.00) complex( 6.00, 1.00)
complex( 2.00, 2.00) complex( 7.00, 2.00) complex(13.00, 2.00)

x1=1.000000 2.000000 3.000000 4.000000
5.000000 6.000000 7.000000 8.000000
5.000000 6.000000 7.000000 8.000000

10.000000 11.000000 12.000000 13.000000
10.000000 11.000000 12.000000 13.000000
14.000000 15.000000 16.000000 17.000000
y1=1.000000 3.000000 6.000000 10.000000
5.000000 11.000000 18.000000 26.000000
5.000000 11.000000 18.000000 26.000000

10.000000 21.000000 33.000000 46.000000
10.000000 21.000000 33.000000 46.000000
14.000000 29.000000 45.000000 62.000000
```

See Also

cumprod(), **product**(), **cproduct**(), **sum**(), **csum**().

curvefit

Synopsis
#include <**numeric.h**>
int curvefit(**double** *a*[&], **double** *x*[&], **double** *y*[&], **void** (**funcs*)(**double**, **double** [ ]), ...
/*[**double** *sig*[ ], **int** *ia*[ ], **double** *covar*[:][:], **double** **chisq*]*/);

Syntax
curvefit(*a, x, y, funcs*)
curvefit(*a, x, y, funcs, sig*)
curvefit(*a, x, y, funcs, sig, ia*)
curvefit(*a, x, y, funcs, sig, ia, covar*)
curvefit(*a, x, y, funcs, sig, ia, covar, chisq*)

Purpose
Fit a set of data points *x* and *y* to a linear combination of specified functions of *x*.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
a An output array which contains coefficients for curve fitting.

x An input array which contains variable values of data set.

y An input array which contains function values of data set.

funcs The routine given by the user that passes the values of base functions evaluated at *x*.

sig An input array which contains individual standard (measurement error) deviations of data set. It can be set to 1 if unknown.

ia An input array. Those components of nonzero entries shall be fitted, and those components of zero entries shall be held at their input values.

covar The covariance matrix of fitting results, describing the fit of data to the model. See algorithm for definition.

chisq Chi-square of fitting.

Description
Given a set of data points in arrays *x* and *y* with individual standard deviations in array *sig*, use χ^2 minimization to fit for some or all of the coefficients in array *a* of a function that depends linearly on, $y = a_i * funcs_i(x)$. The fitted coefficients are passed by array *a*. The program can also pass χ^2 and covariance

matrix *covar*. If the values for array argument *ia* are constants, the corresponding components of the covariance matrix will be zero. The user supplies a routine *funcs*(*x*,*func*) that passes the function values evaluated at *x* by the array *func*.

Algorithm
The general form of the fitting formula is

$$y(x) \quad = \quad \sum_{k=1}^{M} a_k f_k(x)$$

where $f_1(x), ..., f_M(x)$ are arbitrary fixed function of *x*, called base functions. $f_k(x)$ can be a nonlinear function of *x*. The coefficients a_k are determined by minimizing chi-square which is defined as

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - \sum_{k=1}^{M} a_k f_k(x_i)}{\sigma_i} \right)^2$$

where σ_i is the standard deviation of the ith data point. If the standard deviations are unknown, they can be set to the constant value $\sigma = 1$. By defining a $n \times m$ matrix α with element (k, j) as

$$\alpha_{kj} = \sum_{i=1}^{N} \frac{f_j(x_i) f_k(x_i)}{\sigma_i^2}$$

The element (k, j) of the covariance matrix can be expressed as

$$cover_{kj} = [\alpha]_{kj}^{-1}$$

Example 1
Fitting of the data points generated by $f(x) = 4\cos(x) + 3\sin(x) + 2e^x + 1$ with uniform random deviation to the base functions $\cos(x)$, $\sin(x)$, e^x and 1. This example uses **curvefit**(*a*, *x*, *y*, *funcs*) only.

```
#include <stdio.h>
#include <numeric.h>
#define NPT 100
#define SPREAD 0.1
#define NTERM 4

void funcs(double x, double func[]) {
    func[0]=cos(x);
    func[1]=sin(x);
    func[2]=exp(x);
    func[3]=1.0;
}
int main(void) {
    int i,j;
    array double a[NTERM],x[NPT],y[NPT],sig[NPT],u[NPT];

    linspace(x, 0, 10);
    urand(u);
    y = 4*cos(x) + 3*sin(x) +2*exp(x) +(array double [NPT])1.0 + SPREAD*u;
```

```
    curvefit(a,x,y,funcs);
    printf("\n%11s\n","parameters");
    for (i=0;i<NTERM;i++)
       printf(" a[%1d] = %f\n", i,a[i]);
}
```

Output1

```
 parameters
 a[0] = 3.990255
 a[1] = 2.994660
 a[2] = 2.000000
 a[3] = 1.051525
```

Example 2
Fitting of the data points generated by $f(x) = 4\cos(x) + 3\sin(x) + 2e^x + 1$ with uniform random deviation to the base functions of $\cos(x), \sin(x), e^x$ and 1. This example uses all the options of arguments of function **curvefit**(). The values of 3 and 1 for coefficients *a*[2] and *a*[0] are fixed, respectively.

```
#include <stdio.h>
#include <numeric.h>

#define NPT 100
#define SPREAD 0.1
#define NTERM 4

void funcs(double x, double func[]) {
    func[0]=cos(x);
    func[1]=sin(x);
    func[2]=exp(x);
    func[3]=1.0;
}

int main(void) {
    int i,j;
    array int ia[NTERM];
    array double a[NTERM],x[NPT],y[NPT],sig[NPT],covar[NTERM][NTERM],u[NPT];
    double chisq;

    linspace(x, 0, 10);
    urand(u);
    y = 4*cos(x) + 3*sin(x) +2*exp(x) +(array double [NPT])1.0 + SPREAD*u;
    sig=(array double[NPT])(SPREAD);

    ia[0] = 0; ia[1] = 1; ia[2] = 0; ia[3] = 1;
    a[0] = 4; a[1] = 3; a[2] = 2; a[3] = 1;

    curvefit(a,x,y,funcs,sig,ia,covar,&chisq);
    printf("\n%11s %21s\n","parameter","uncertainty");
    for (i=0;i<NTERM;i++)
        printf("  a[%1d] = %8.6f %12.6f\n",i,a[i],sqrt(covar[i][i]));
    printf("chi-square = %12f\n",chisq);
    printf("full covariance matrix\n");
    printf("%12f",covar);
}
```

Output 2

```
  parameter           uncertainty
```

```
  a[0] = 4.000000     0.000000
  a[1] = 2.994009     0.015044
  a[2] = 2.000000     0.000000
  a[3] = 1.052622     0.010357
chi-square =     7.538080
full covariance matrix
    0.000000     0.000000     0.000000     0.000000
    0.000000     0.000226     0.000000    -0.000041
    0.000000     0.000000     0.000000     0.000000
    0.000000    -0.000041     0.000000     0.000107
```

See Also

polyfit(), **std**().

References

William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannerya, *Numeric Recipes in C*, Second Edition, Cambridge University Press, 1997.

deconv

Synopsis
#include <**numeric.h**>
int deconv(array double complex *u*[&], **array double complex** *v*[&], **array double complex** *q*[&], ...
/* [**array double complex** *r*[&]] */);

Syntax
deconv(*u*, *v*, *q*)
deconv(*u*, *v*, *q*, *r*)

Purpose
One-dimensional Discrete Fourier Transform (DFT) based deconvolution.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
u A one-dimensional array of size n. It contains data used for deconvolution.

v A one-dimensional array of size m. It contains data used for deconvolution.

q A one-dimensional array of size $(n - m + 1)$. It contains the result of deconvolution of *u* and *v*.

r A one-dimensional array of size n. It contains the remainder item if user specified.

Description
The input arrays *u* and *v* can be of any supported arithmetic data type and sizes n and m. Conversion of the data to **double complex** type is performed internally. If both *u* and *v* are **real** data type, the deconvolution result is a one-dimensional **real array** *q* of size $(n + m - 1)$ and the remaining item is a **real array** *r* if specified. If either one of *u* and *v* is a **complex** type, then the result **array** *q* and remained item *r* will be a **complex** data type.
If *u* and *v* are vectors of polynomial coefficients, deconvolution of *u* and *v* is equivalent to the division the two polynomials.

Algorithm
According to the theorem of convolution, if $X(f)$ and $Y(f)$ are Fourier transforms of $x(t)$ and $y(t)$, that is,

$$x(t) \Longleftrightarrow X(f) \qquad \text{and} \qquad y(t) \Longleftrightarrow Y(f)$$

then

$$x * y \Longleftrightarrow X(f)Y(f)$$

The theorem of convolution of discrete convolution is described as,
if $X_d(f)$ and $Y_d(f)$ are discrete Fourier transforms of sequences $x[nT]$ and $y[nT]$, that is,

$$x[nT] \Longleftrightarrow X_d(f) \qquad \text{and} \qquad y[nT] \Longleftrightarrow Y_d(f)$$

then

$$x[nT] * y[nT] \Longleftrightarrow X_f(f)Y_d(f)$$

where T is the sample interval and n is an index used in the summation.
In function **deconv**(), the convolution result *u* is given and one sequence of convolution data *v* is known. Based on the theorem of discrete convolution, DFT can be used to compute discrete deconvolution. The procedure is first to compute the DFT of $u[nT]$ and $v[nT]$ and then use the theorem to compute

$$q[nT] + r[nT] = F_d^{-1}[U_d(f)]/F_d^{-1}[V_d(f)]$$

where $U_d(f)$ and $V_d(f)$ are DFT of sequences $u[nT]$ and $v[nT]$, respectively. In the algorithm the long division of two polynomial is used to get the result. The quotient is the convolution sequence $q[nT]$ and the remainder is $r[nT]$.

Example
There are two polynomial *U* and *V* whose coefficients *u* and *v* are $[6, 19, 32, 35, 58, 71, 42]$ and $[6, 7]$ respectively. The deconvolution of *u* and *v* is equivalent to the division of polynomial *U* by polynomial *V*.

$$\begin{aligned} U &= 6 * t^6 + 19 * t^5 + 32 * t^4 + 45 * t^3 + 58 * t^2 + 71 * t + 42 \\ V &= 6 * t + 7 \end{aligned}$$

then U/V equals

$$Q = t^5 + 2 * t^4 + 3 * t^3 + 4 * t^2 + 5 * t + 6$$

```
#include <stdio.h>
#include <numeric.h>

#define N 7         /* u array size */
#define M 2         /* response function dimension  */
#define N2 N-M+1

int main() {
   int i,j,n,m;
   array double u[N] = {6,19,32,45,58,71,42},
                v[M] = {6,7};
   array double q[N2], r[N];

   printf("u=%f\n",u);
   printf("v=%f\n",v);

   deconv(u,v,q);  /* real v, not specify remaind result */
   deconv(u,v,q,r); /* real v, specified remaind result */

   printf("q=%f\n",q);
   printf("r=%f\n",r);
}
```

Output

```
u=6.000000 19.000000 32.000000 45.000000 58.000000 71.000000 42.000000

v=6.000000 7.000000

q=1.000000 2.000000 3.000000 4.000000 5.000000 6.000000

r=0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 -0.000000
```

See Also

conv(), **conv2**(), **filter**().

References

Nussbaumer, H. J, *Fast Fourier Transform and Convolution Algorithms*, New York: Springer-Verlay, 1982.

derivative

Synopsis
#include <**numeric.h**>
double derivative(**double** (**func*)(**double**), **double** *x*, ... /* [**double** &*err*], [**double** *h*] */);

Syntax
derivative(*func*, *x*)
derivative(*func*, *x*, &*err*)
derivative(*func*, *x*, &*err*, *h*)

Purpose
Calculate numerical derivative of a function at a given point.

Return Value
This function returns a derivative value of function *func* at point *x* in **double** data type.

Parameters
func The function taking derivative.

x A point at which the derivative is evaluated.

err An estimated error in the derivative will return as *err* if this argument is passed.

h An initial estimated step size. It does not have to be small, but it should be an increment in *x* over which *func* changes substantially. If the user does not specify this value, the value of $0.02 * x$ or 0.0001 (if *x* < 0.0001) is used by default.

Description
The function taken derivative, *func*, is specified as a pointer to a function that takes a **double** as an argument and returns a **double** value at *x*. The returned value is the derivative of the function at point *x*. The optional argument *err* contains the estimate of the error in the calculation of derivative. It helps the user to estimate the result. If the optional argument *h* is given, calculation of the derivative uses the initial step size of *h*. Otherwise it uses the default value.

Algorithm
The symmetrized form of the definition of the derivate is defined as the limit

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

when $h \to 0$. For this formula, the truncation and roundoff errors are $e_t \sim h^2 f'''$ and $e_r \sim \epsilon_f |f(x)/h|$,

respectively. The optimal choice of h is

$$h \sim (\frac{\epsilon_f f}{f'''})^{1/3} \sim (\epsilon_f)^{1/3} x_c$$

and the relative error is

$$(e_r + e_t)/|f'| \sim (\epsilon_f)^{2/3} f^{2/3} (f''')^{1/3}/f' \sim (\epsilon_f)^{2/3}$$

Example

Calculate the derivative of function $x\sin(x)$ at point $x = 2.5$

$$\frac{d}{dx} x\sin(x)|_{x=2.5} = \{\sin(x) - x\cos(x)\}|_{x=2.5} = -1.404387$$

```
#include <stdio.h>
#include <math.h>
#include <numeric.h>

double func(double x) {
    return x*sin(x);
}

int main() {
    double yprime, func(double), x, h, err;

    x = 2.5;
    yprime = derivative(func, x);
    /* derivative of func = x*sin(x) is sin(x)+x*cos(x) */
    printf("yprime = %f\n", sin(x)+x*cos(x));
    printf("derivative(func, x) = %f\n", yprime);
    yprime  = derivative(func, x, &err);
    printf("derivative(func, x, &err) = %f, err = %f\n", yprime, err);
    h = 0.1; // default h = 0.001
    yprime  = derivative(func, x, &err, h);
    printf("derivative(func, x, &err, h) = %f, err = %f\n", yprime, err);
}
```

Output

```
yprime = -1.404387
derivative(func, x) = -1.404387
derivative(func, x, &err) = -1.404387, err = 0.000000
derivative(func, x, &err, h) = -1.404387, err = 0.000000
```

see also

derivatives(), **integral1**(), **integral2**(), **integral3**(), **integration2**(), **integration3**().

References

Ridder, C. J. F., *Advances in Engineering Software*, 1982, vol. 4, no. 2, pp. 75-76. [2].

derivatives

Synopsis
#include <**numeric.h**>
array double derivatives(**double** (**func*)(**double**), **double** *x*[&], ... /* [**double** &*err*], [**double** *h*] */)[:];

Syntax
derivative(*func*, *x*)
derivative(*func*, *x*, &*err*)
derivative(*func*, *x*, &*err*, *h*)

Purpose
Calculate derivatives of a function numerically at multiple points.

Return Value
This function returns an **array** of derivative values of function *func* at points *x*[&].

Parameters
func Function its derivatives are calculated.

x A series of points in which derivatives are calculated.

err An estimated error in the derivative which will return as *err* if the user specified.

h An initial estimated step size. It does not have to be small, but rather should be an increment in *x* over which *func* changes substantially. If the user do not specify this value, the value of $0.02 * x$ or 0.0001 (if $x < 0.0001$) is used by default.

Description
The function taking derivative, *func*, is specified as a pointer to a function that takes an **array** of **double** data as an argument and returns an **array double** values at *x*[&]. The values of *x* can be of any real type. Conversion of the data to **double** is performed internally. The return value is an **array** of derivatives of the function at points specified by **array** *x*. If the optional argument *h* is specified, which shall be **double** data type, the derivative uses initial step size of *h*. Otherwise it uses the default value. Optional argument *err* shall be a **double** data and it means the estimate of the error in the derivate. This helps the user to estimate the result.

Algorithm
Using a for-loop to evaluate the derivative values specified in **array** *x* and in the loop using the same algorithm as **derivative**().

Example

Take the derivatives of function

$$f(x) = x * \sin(x) * \cos(x)$$

at 36 different positions meanly spaced between 0 and 2π.
Analytically, the first derivative of this function is

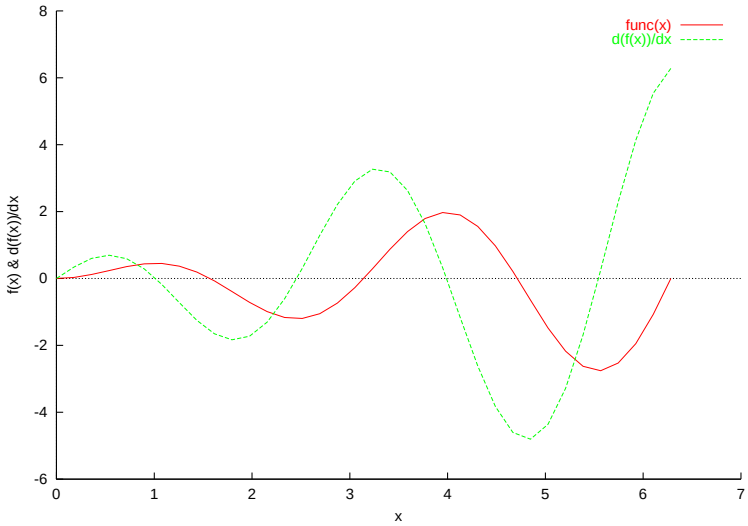
$$f'(x) = 0.5 * \sin(2x) + x\cos(2x)$$

```
#include <stdio.h>
#include <math.h>
#include <chplot.h>
#include <numeric.h>

#define N 36

double func(double x) {
   return x*sin(x)*cos(x);
}

int main() {
   array double x[N], y[N], yprime[N];
   double err, h;
   int i;
   class CPlot plot;

   linspace(x, 0, 2*M_PI);
   printf("       x      derivative() value       Cal. value\n");
//   /* Specify the initial step h and return estimate error */
//   yprime =derivatives(func,x,&err,0.8);
   yprime =derivatives(func,x);    /* use default step value and do not return err */

   fevalarray(y,func,x);

   for (i=0; i<2; i++)
      printf("%12.6f    %12.6f        %12.6f\n",x[i],yprime[i],
             (0.5*sin(2*x[i])+x[i]*cos(2*x[i])));
   printf("     --------     ---------            --------\n");
   for (i=34; i<36; i++)
      printf("%12.6f    %12.6f        %12.6f\n",x[i],yprime[i],
             (0.5*sin(2*x[i])+x[i]*cos(2*x[i])));
   plot.label(PLOT_AXIS_X,"x");
   plot.label(PLOT_AXIS_Y,"f(x) & d(f(x))/dx");
   plot.data2D(x,y);
   plot.data2D(x,yprime);
   plot.legend("f(x)",0);
   plot.legend("d(f(x))/dx",1);
   plot.plotting();
}
```

Output

```
       x derivative() value  Cal. value
    0.000000     0.000000     0.000000
    0.179520     0.343760     0.343760
```

```
 -------- -----------------
6.103666     5.538781     5.538781
6.283186     6.283187     6.283187
```



see also

derivative(), **integral1**(), **integral2**(), **integral3**(), **integration2**(), **integration3**().

References

Ridder, C. J. F., *Advances in Engineering Software*, 1982, vol. 4, no. 2, pp. 75-76.

determinant

Synopsis
#include <**numeric.h**>
double determinant(**array double** *a*[&][&]);

Purpose
Calculate the determinant of a matrix.

Return Value
This function returns the determinant of matrix *a*.

Parameters
a The input matrix.

Description
The function **determinant**() returns the determinant of matrix *a*. If the input matrix is not a square matrix, the function will return NaN.

Example
Calculate the determinant of matrices with different data type.

```
#include <numeric.h>
int main() {
    array double a[2][2] = {2, 4,
                            3, 7};
    /* a2 is an ill-detition matrix  */
    array double a2[2][2] = {2, 4,
                             2.001, 4.0001};
    /* a3 is singular */
    array double a3[2][2] = {2, 4,
                             4, 8};
    array float b[2][2] = {2, 4,
                           3, 7};
    array double c[3][3] = {-1,5,6,
                             3,-6,1,
                             6,8, 9} ; /* n-by-n matrix */
    array double d[3][3] = {2,1,-2,
                            4,-1,2,
                            2,-1,1} ; /* n-by-n matrix */

    double det;

    det = determinant(a);
    printf("determinant(a) = %g\n", det);
    det = determinant(a2);
    printf("determinant(a2) = %g\n", det);
    det = determinant(a3);
    printf("determinant(a3) = %g\n", det);
    det = determinant(b);
    printf("determinant(b) = %g\n", det);
```

```
    det = determinant(c);
    printf("determinant(c) = %g\n", det);
    det = determinant(d);
    printf("determinant(d) = %g\n", det);
}
```

Output

```
determinant(a) = 2
determinant(a2) = -0.0038
determinant(a3) = -0
determinant(b) = 2
determinant(c) = 317
determinant(d) = 6
```

See Also

cdeterminant(), **inverse**(), **diagonal**(), **ludecomp**(), **rcondnum**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

diagonal

Synopsis
#include <**numeric.h**>
array double diagonal(**array double** *a*[&][&], ... /* [**int** *k*] */)[:];

Syntax
diagonal(*a*);
diagonal(*a, k*);

Purpose
Form a vector with diagonals of a matrix.

Return Value
This function returns a column vector formed from the elements of the *k*th diagonal of the matrix *a*. By default, *k* is 0.

Parameters
a An input matrix.

k An input integer indicating which element the vector is formed from.

Description
The function **diagonal**() produces a colum vector formed from the elements of the *k*th diagonal of the matrix *a*.

Example1

```
#include <numeric.h>
int main() {
    array double a[3][3] = {1, 2, 3,
                            4, 5, 6,
                            7, 8, 9};
    int n =3, k = 1;
    array double d[n], d2[n-(abs(k))];

    d = diagonal(a);
    printf("diagonal(a) = %f\n", d);

    d2 = diagonal(a, k);
    printf("diagonal(a, 1) = %f\n", d2);

    k = -1;
    d2 = diagonal(a, k);
    printf("diagonal(a, -1) = %f\n", d2);
}
```

Output1

```
diagonal(a) = 1.000000 5.000000 9.000000

diagonal(a, 1) = 2.000000 6.000000

diagonal(a, -1) = 4.000000 8.000000
```

Example2

```
#include <numeric.h>
int main() {
    array double a[4][3] =  {1, 2, 3,
                             4, 5, 6,
                             7, 8, 9,
                             4, 4, 4};
    int m=4, n = 3, k1 = 1, k2 = -1;
    array double d[min(m, n)];
    array double d1[min(m, n-k1)];
    array double d2[min(m+k2, n)];

    d = diagonal(a);
    printf("diagonal(a) = %f\n", d);

    d1 = diagonal(a, k1);
    printf("diagonal(a, 1) = %f\n", d1);

    d2 = diagonal(a, k2);
    printf("diagonal(a, -1) = %f\n", d2);
}
```

Output2

```
diagonal(a) = 1.000000 5.000000 9.000000

diagonal(a, 1) = 2.000000 6.000000

diagonal(a, -1) = 4.000000 8.000000 4.000000
```

Example3

```
#include <numeric.h>
int main() {
    array double a[3][4] =  {1, 2, 3, 4,
                             5, 6, 7, 8,
                             9, 4, 4, 4};
    int m=3, n = 4, k1 = 1, k2 = -1;
    array double d[min(m, n)];
    array double d1[min(m, n-k1)];
    array double d2[min(m+k2, n)];

    d = diagonal(a);
    printf("diagonal(a) = %f\n", d);

    d1 = diagonal(a, k1);
    printf("diagonal(a, 1) = %f\n", d1);

    d2 = diagonal(a, k2);
    printf("diagonal(a, -1) = %f\n", d2);
}
```

Output3

```
diagonal(a) = 1.000000 6.000000 4.000000

diagonal(a, 1) = 2.000000 7.000000 4.000000

diagonal(a, -1) = 5.000000 4.000000
```

See Also
diagonalmatrix().
cdiagonal(), **cdiagonalmatrix**().

References

diagonalmatrix

Synopsis
#include <**numeric.h**>
array double diagonalmatrix(array double *v*[&], ... /* [int k] */)[:][:];

Syntax
diagonalmatrix(*v*)
diagonalmatrix(*v, k*)

Purpose
Form a diagonal matrix.

Return Value
This function returns a square matrix with specified diagonal elements.

Parameters
v An input vector.

k An input integer indicating specified diagonal elements of the matrix.

Description
The function **diagonalmatrix()** returns a square matrix of order $n + abs(k)$, with the elements of *v* on the *k*th diagonal. $k = 0$ represents the main diagonal, $k > 0$ above the main diagonal, and $k < 0$ below the main diagonal.

Example

```
#include <numeric.h>
int main() {
    array double v[3] = {1,2,3};
    int n = 3, k;
    array double a[n][n];

    a = diagonalmatrix(v);
    printf("diagonal matrix a =\n%f\n", a);

    k = 0;
    a = diagonalmatrix(v,k);
    printf("diagonalmatrix(a, 0) =\n%f\n", a);

    k = 1;
    array double a2[n+abs(k)][n+abs(k)];
    a2 = diagonalmatrix(v,k);
    printf("diagonalmatrix(a2, 1) =\n%f\n", a2);

    k = -1;
    array double a3[n+abs(k)][n+abs(k)];
    a3 = diagonalmatrix(v,k);
```

```
    printf("diagonalmatrix(a3, -1) =\n%f\n", a3);
}
```

Output

```
diagonal matrix a =
1.000000 0.000000 0.000000
0.000000 2.000000 0.000000
0.000000 0.000000 3.000000

diagonalmatrix(a, 0) =
1.000000 0.000000 0.000000
0.000000 2.000000 0.000000
0.000000 0.000000 3.000000

diagonalmatrix(a2, 1) =
0.000000 1.000000 0.000000 0.000000
0.000000 0.000000 2.000000 0.000000
0.000000 0.000000 0.000000 3.000000
0.000000 0.000000 0.000000 0.000000

diagonalmatrix(a3, -1) =
0.000000 0.000000 0.000000 0.000000
1.000000 0.000000 0.000000 0.000000
0.000000 2.000000 0.000000 0.000000
0.000000 0.000000 3.000000 0.000000
```

See Also

cdiagonalmatrix(), **diagonal**() **cdiagonal**().

References

difference

Synopsis
#include <**numeric.h**>
array double difference(**array double** *a*[&])[:];

Purpose
Calculate differences between the adjacent elements of an array.

Return Value
This function returns the array of differences of adjacent elements of array *a*.

Parameters
a The input array.

Description
The function **difference**() calculates the differences of adjacent elements of an array. The input array can be of any arithmetic data types.

Example
Calculate the differences of the adjacent elements of the arrays with different data types.

```
#include <numeric.h>
#define N 6
int main() {
    array double a[N] = {1, 2, 10, 4, 5, 6};
    array float f[N] = {1, 2, 10, 4, 5, 6};
    array double d[N-1];

    d = difference(a);
    printf("difference(a) = %f\n", d);
    d = difference(f);
    printf("difference(f) = %f\n", d);
}
```

Output

```
difference(a) = 1.000000 8.000000 -6.000000 1.000000 1.000000

difference(f) = 1.000000 8.000000 -6.000000 1.000000 1.000000
```

See Also
derivative(), **derivatives**(), **sum**(), **product**().

References

dot

Synopsis
#include <**numeric.h**>
double dot(**array double** *a*[&], **array double** *b*[&]);

Purpose
Calculate the dot product of a vector.

Return Value
This function returns the dot product of two vectors.

Parameters
a An input one-dimensional array.

b Another input one-dimensional array.

Description
The function calculates the dot product of two vectors. The number of elements of two vectors should be the same. Otherwise the function returns NaN.

Example

```
#include <numeric.h>
int main() {
    array double a[4] = {1, 2, 3, 4};
    array double b[ ] = {1, 2, 3, 4};
    double dotprod;

    dotprod = dot(a, b);
    printf("dot(a,b) = %f\n", dotprod);
}
```

Output

```
dot(a,b) = 30.000000
```

See Also
cross().

References

eigen

Synopsis
#include <**numeric.h**>
int eigen(... /* **array double** [**complex**] *a*[:][:], **array double** [**complex**] *evalues*[:],
array double [**complex**] *evectors*[:][:], **string_t** *mode*] */);

Syntax
eigen(a, *evalues*); eigen(a, *evalues*, *evectors*); eigen(a, *evalues*, *evectors*, `mode`);
Purpose
Find eigenvalues and eigenvectors.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
a Input square matrix for which the eigenvalues and eigenvectors are calculated.

evalues Output one-dimensional array which contains the calculated eigenvalues.

evectors Output square matrix which contains the calculated eigenvectors.

mode The optional **string_t** *mode* used to specify if using a preliminary balancing step before the calculation of eigenvalues and eigenvectors.

Description
This function calculates the eigenvalues and eigenvectors of n-by-n matrix *a* so that $a*v = \lambda*v$, where λ and v are the vector of eigenvalues and eigenvectors, respectively. The computed eigenvectors are normalized so that the norm of each eigenvector equals 1. *mode* is used to specify if a preliminary balancing step before the calculation is performed or not. Generally, balancing improves the condition of the input matrix, enabling more accurate computation of the eigenvalues and eigenvectors. But, it also may lead to incorrect eigenvectors in special cases. By default, a preliminary balancing step is taken.
Example1
This example calculates the eigenvalues and eigenvectors of a non-symmetrical matrix. The eigenvalues of this matrix is real number. The formula $a * v - evalues * v$ is calculated to verify the results.

```
#include <numeric.h>
#define N 5

int main() {
array double a[5][5] = {2,      30,    100,  1000, 10000,
                        0.3,    1,     10,   100,  1000,
                        0.04,   0.1,   1,    10,   100,
                        0.001,  0.01,  0.1,  1,    10,
                        0.0002, 0.003, 0.01, 0.1,  1};
     array double evalues[N];
     array double evectors[N][N];
     array double *v, vv[N][N];
```

```
    int i, status;

    eigen(a, evalues, evectors);
    printf("eigennvalues of a =\n%e\n", evalues);
    printf("eigenvectors of a =\n%e\n", evectors);

    eigen(a, evalues, evectors, "nobalance");
    printf("no balance: eigennvalues of a =\n%e\n", evalues);
    printf("no balance: eigenvectors of a =\n%e\n", evectors);

    vv = transpose(evectors);
      for(i=0; i<N; i++) {
         v = (array double [:])(double [N])&vv[i][0];
         printf("A*v-evalues[i]*v = %f\n", a*v-evalues[i]*v);
      }

}
```

Output1

```
eigennvalues of a =
7.327477e+00 -1.508424e+00 1.809473e-01 -1.555781e-15 2.032652e-16

eigenvectors of a =
-9.959858e-01 -9.945522e-01 -9.966546e-01 -7.564396e-14 4.879875e-14
-8.892060e-02 1.028639e-01 -5.396722e-02 -9.891903e-15 -2.815928e-14
-1.025131e-02 1.687971e-02 -6.047655e-02 9.948856e-01 8.268104e-01
-6.173564e-04 -2.900250e-04 1.047629e-02 -1.010080e-01 5.588107e-01
-9.959858e-05 -9.945522e-05 -9.966546e-05 1.519397e-04 -6.414918e-02

no balance: eigennvalues of a =
7.327477e+00 1.809473e-01 -1.508424e+00 1.227397e-15 -4.649855e-16

no balance: eigenvectors of a =
9.959858e-01 9.966546e-01 -9.945522e-01 1.586027e-13 -2.911653e-16
8.892060e-02 5.396722e-02 1.028639e-01 2.746367e-15 -1.029631e-14
1.025131e-02 6.047655e-02 1.687971e-02 9.842796e-01 9.714797e-01
6.173564e-04 -1.047629e-02 -2.900250e-04 -1.764452e-01 -2.367118e-01
9.959858e-05 9.966546e-05 -9.945522e-05 7.801720e-03 1.395638e-02

A*v-evalues[i]*v = -0.000000 0.000000 0.000000 0.000000 -0.000000

A*v-evalues[i]*v = 0.000000 0.000000 0.000000 0.000000 -0.000000

A*v-evalues[i]*v = 0.000000 0.000000 0.000000 0.000000 0.000000

A*v-evalues[i]*v = 0.000000 0.000000 0.000000 0.000000 0.000000

A*v-evalues[i]*v = -0.000000 0.000000 0.000000 0.000000 -0.000000
```

Example2
The eigenvalues and eigenvectors of symmetrical and non-symmetrical matrices are calculated. The symmetrical matrix has real eigenvalues and a non-symmetrical matrix has real or complex eigenvalues.

```
/* File: eigen_1.ch
   Find eigenvalues and eigenvectors */
#include <numeric.h>
int main() {
```

```
     /* eigenvalues of a symmtrical matrix are always real numbers */
     array double a[3][3] = {0.8, 0.2, 0.1,
                             0.2, 0.7, 0.3,
                             0.1, 0.3, 0.6};
     /* eigenvalues of a non-symmtrical matrix can be either real numbers or
        complex numbers */
     /* this non-symmtrical matrix has real eigenvalues */
     array double a2[3][3] = {0.8, 0.2, 0.1,
                             0.1, 0.7, 0.3,
                             0.1, 0.1, 0.6};
     /* this non-symmtrical matrix has complex eigenvalues and eigenvectors*/
     array double a3[3][3] = {3, 9, 23,
                              2, 2, 1,
                              -7, 1, -9};
     array double evalues[3],evectors[3][3];
     array double complex zevalues[3], zevectors[3][3];

     eigen(a, evalues);
     printf("evalues =\n%f\n", evalues);
     eigen(a2, evalues);
     printf("evalues =\n%f\n", evalues);
     eigen(a2, zevalues, evectors);
     printf("zevalues =\n%f\n", zevalues);
     printf("evectors =\n%f\n", evectors);
     eigen(a3, evalues);
     printf("evalues =\n%f\n", evalues);
     eigen(a3, zevalues, zevectors);
     printf("evalues =\n%f\n", zevalues);
     printf("evectors =\n%f\n", zevectors);
}
```

Output2

```
evalues =
1.108807 0.652624 0.338569

evalues =
1.000000 0.600000 0.500000

zevalues =
complex(1.000000,0.000000) complex(0.600000,0.000000) complex(0.500000,0.000000)


evectors =
-0.744845 -0.707107 0.408248
-0.579324 0.707107 -0.816497
-0.331042 0.000000 0.408248

evalues =
NaN NaN 3.241729

evalues =
complex(-3.620864,10.647303) complex(-3.620864,-10.647303) complex(3.241729,0.000000)


evectors =
complex(0.854461,0.000000) complex(0.854461,0.000000) complex(0.604364,0.000000)

complex(-0.024440,-0.125397) complex(-0.024440,0.125397) complex(0.744064,0.000000)
```

```
complex(-0.236405,0.444621) complex(-0.236405,-0.444621) complex(-0.284803,0.000000)
```

Example3
The eigenvalues and eigenvectors of real and complex matrices are calculated in this example.

```
#include <numeric.h>
int main() {
     array double complex a[3][3] = {0.8, 0.2, 0.1,
                                     0.1, 0.7, 0.3,
                                     0.1, 0.1, 0.6};

     array double complex a2[3][3] = {complex(0.8,-1), 0.2, 0.1,
                                                0.1, 0.7, 0.3,
                                                0.1, 0.1, 0.6};
    /* this non-symmtrical matrix has complex eigenvalues and eigenvectors*/
    array double complex a3[3][3] = {3, 9, 23,
                                    2, 2, 1,
                                    -7, 1, -9};


     array double complex evalues[3];
     array double complex evectors[3][3];

     eigen(a, evalues, evectors);
     printf("eigenvalues of a =\n%f\n", evalues);
     printf("eigenvectors of a =\n%f\n", evectors);

     eigen(a2, evalues, evectors);
     printf("eigenvalues of a =\n%f\n", evalues);
     printf("eigenvectors of a =\n%f\n", evectors);

     eigen(a3, evalues, evectors);
     printf("eigenvalues of a =\n%f\n", evalues);
     printf("eigenvectors of a =\n%f\n", evectors);
}
```

Output3

```
eigenvalues of a =
complex(1.000000,0.000000) complex(0.600000,0.000000) complex(0.500000,0.000000)


eigenvectors of a =
complex(0.744845,0.000000) complex(-0.707107,0.000000) complex(-0.408248,0.000000)

complex(0.579324,0.000000) complex(0.707107,0.000000) complex(0.816497,0.000000)

complex(0.331042,0.000000) complex(0.000000,0.000000) complex(-0.408248,0.000000)


eigenvalues of a =
complex(0.796911,-0.968453) complex(0.832118,-0.037024) complex(0.470970,0.005477)


eigenvectors of a =
complex(0.989164,0.000000) complex(0.003912,-0.225655) complex(-0.025122,-0.090331)

complex(-0.019749,0.106886) complex(0.891481,0.000000) complex(0.794571,0.000000)
```

```
complex(0.008946,0.098280) complex(0.391299,-0.034802) complex(-0.598227,0.044616)


eigenvalues of a =
complex(-3.620864,10.647303) complex(-3.620864,-10.647303) complex(3.241729,-0.000000)


eigenvectors of a =
complex(0.854461,0.000000) complex(0.854461,0.000000) complex(0.604364,0.000000)

complex(-0.024440,-0.125397) complex(-0.024440,0.125397) complex(0.744064,0.000000)

complex(-0.236405,0.444621) complex(-0.236405,-0.444621) complex(-0.284803,-0.000000)
```

See Also

balance(), **choldecomp**(), hessdecomp().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

eigensystem

Synopsis
#include <**numeric.h**>
int eigensystem(**array double complex** *evalues*[&], **array double complex** *evectors*[&][&],
array double complex *a*[&][&], ... /* [**string_t** *mode*] */);

Syntax
eigensystem(*evalues*, NULL, *a*); eigensystem(*evalues*, *evectors*, *a*);
Purpose
Find eigenvalues and eigenvectors. This function is obsolte. Use function **eigen**().

Return Value
This function returns 0 on success and -1 on failure.

Parameters
evalues Output one-dimensional array which contains the calculated eigenvalues.

evectors Output square matrix which contains the calculated eigenvectors.

a Input square matrix for which the eigenvalues and eigenvectors are calculated.

mode The optional **string_t** *mode* used to specify if using a preliminary balancing step before the calculation of eigenvalues and eigenvectors.

Description
This function calculates the eigenvalues and eigenvectors of n-by-n matrix *a* so that $a*v = \lambda*v$, where λ and v are the vector of eigenvalues and eigenvectors, respectively. The computed eigenvectors are normalized so that the norm of each eigenvector equals 1. *mode* is used to specify if a preliminary balancing step before the calculation is performed or not. Generally, balancing improves the condition of the input matrix, enabling more accurate computation of the eigenvalues and eigenvectors. But, it also may lead to incorrect eigenvectors in special cases. By default, a preliminary balancing step is taken.

See Also
eigen().

References
E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

expm

Synopsis
#include <**numeric.h**>
int expm(**array double complex** *y[&][&]*, **array double complex** *x[&][&]*);

Syntax
expm(*y*, *x*)

Purpose
Computes the matrix exponential.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Input square matrix. It contains data to be calculated.

y Output square matrix which contains data of the result of *x* exponential.

Description
This function computes the matrix exponential of the input matrix *x* using a scaling and squaring algorithm with a Pade approximation. If any of the eigenvalues of input matrix *x* are positive, the output matrix *y* is a real matrix. If *x* has nonpositive eigenvalues, the results *y* shall be a complex matrix.

Example
Calculation of exponentials of real and complex matrix.

```
#include <numeric.h>
int main() {
   array double x[3][3]={1,2,3,
                         3,4,5,
                         6,7,8};
   array double complex zx[3][3]={complex(1,1),complex(2,2),0,
                         3,complex(4,1),complex(2,5),
                         0,0,0};
   array double complex zy[3][3];
   array double y[3][3];

   expm(y,x);
   printf("x = \n%f",x);
   printf("y = \n%f",y);
   printf("\n");

   expm(zy,zx);
   printf("zx = \n%5.1f",zx);
   printf("zy = \n%5.1f",zy);
}
```

Output

```
x =
1.000000 2.000000 3.000000
3.000000 4.000000 5.000000
6.000000 7.000000 8.000000
y =
157372.953093 200034.605129 242697.257164
290995.910241 369883.552084 448769.193928
491431.845963 624654.472518 757878.099072

zx =
complex(  1.0,  1.0) complex(  2.0,  2.0) complex(  0.0,  0.0)
complex(  3.0,  0.0) complex(  4.0,  1.0) complex(  2.0,  5.0)
complex(  0.0,  0.0) complex(  0.0,  0.0) complex(  0.0,  0.0)
zy =
complex(-44.9, 56.8) complex(-87.5, 70.9) complex(-101.5,-18.9)
complex(-12.5,118.7) complex(-57.4,175.5) complex(-155.5, 67.8)
complex(  0.0,  0.0) complex(  0.0,  0.0) complex(  1.0,  0.0)
```

See Also

logm(), **funm**(), **cfunm**(), **sqrtm**().

References

G. H. Golub, C. F. Van Loan, Matrix Computations Third edition, The Johns Hopkins University Press, 1996

factorial

Synopsis
#include <**numeric.h**>
unsigned long long factorial(**unsigned int** *n*);

Purpose
Calculate factorial of an unsigned integer.

Return Value
This function returns factorial of integer *n*.

Parameters
n Input unsigned integer.

Description
The function **factorial** calculates the product of all the integers from 1 to *n*. The factorial is defined as

$$f = n!$$

Example

```
#include <numeric.h>
int main() {
    unsigned long long f;

    f = factorial(3);
    printf("factorial(3) = %llu\n", factorial(3));
}
```

Output

```
factorial(3) = 6
```

See Also
combination().

References

fevalarray

Synopsis
#include <**numeric.h**>
int fevalarray(**array double** &*y*, **double** (**func*)(**double** *x*), **array double** &*x*, ... /* [**array int** &*mask*, **double** *value*] */);

Syntax
fevalarray(*y, func, x*)
fevalarray(*y, func, x, mask, value*)

Purpose
Array function evaluation.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
y An output array which contains the result of function evaluation.

x An input array at which the function values will be evaluated.

func A pointer to function given by the user.

mask An input array of integer. If the value of an element of array *mask* is 1, the function evaluation will be applied to that element. If the value of an element of array is 0, the value from the optional fifth argument *value* will be used for that element.

value The default value used to replace the function evaluation when the entry of *mask* is 0.

Description
This function evaluates a user function applied to each element of the input array *x*. The input array can be of any arithmetic data type and dimension.

Example
Evaluation of function $f(x) = x^2$ for arrays of different dimensions.

```
#include <numeric.h>
#define N 4
#define M 5

double func(double x) {
   return x*x;
}

int main() {
  array double y1[N],    x1[N];
```

```
  array float y2[N][M], x2[N][M]; /* diff data types, diff dim */
  array int mask1[N] = {1, 0, 1, 0};

  linspace(x1, 1, N);
  fevalarray(y1, func, x1);
  printf("y1 = %f\n", y1);

  linspace(x1, 1, N);
  fevalarray(y1, func, x1, mask1, -1);
  printf("y1 = %f\n", y1);


  linspace(x2, 1, N*M);
  fevalarray(y2, func, x2);
  printf("y2 = %f\n", y2);
}
```

Output

```
y1 = 1.000000 4.000000 9.000000 16.000000

y1 = 1.000000 -1.000000 9.000000 -1.000000

y2 = 1.000000 4.000000 9.000000 16.000000 25.000000
36.000000 49.000000 64.000000 81.000000 100.000000
121.000000 144.000000 169.000000 196.000000 225.000000
256.000000 289.000000 324.000000 361.000000 400.000000
```

See Also

cfevalarray(), **streval**().

References

fft

Synopsis
#include <**numeric.h**>
int fft(array double complex &*y*, **array double complex** &*x*, ... /* [**int** *n* [**int** *dim*[&]]] */);

Syntax
fft(*y*, *x*)
fft(*y*, *x*, *n*)
fft(*y*, *x*, *dim*);

Purpose
N-dimensional Fast Fourier Transform (FFT) calculation.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
y Result of FFT, the same dimension and size **array** as *x*.

x An n-dimensional array of data used for FFT.

n Optional argument, user specified FFT points for one-dimensional data.

dim A one-dimensional array of optional arguments with **int** type. It contains user specified FFT points. Each element corresponds to data in that dimension. For example, there is a three-dimensional data *x[m][n][l]*. The FFT points of the array is specified by m, n, l. Then the array *dim* is given values of $\dim[0] = m, \dim[1] = n$ and $\dim[2] = l$.

Description
The multi-dimensional (maximum dimension is three) array *x* can be of any supported arithmetic data type and size. Conversion of the data to double complex is performed internally. Array *y* with the same dimension as array *x* contains the result of the fast Fourier transform. The optional argument *n* of int type is used to specify the number of points for FFT of one-dimensional data For multi-dimensional data, The optional array argument *dim* of int type. contains the value for user specified FFT points. The dimensions of the input array *x* are contained in array *dim*. If no optional argument is passed, the number of FFT points is obtained from the input array *x*.

Algorithm
Mixed-radix Fast Fourier Transform algorithm developed by R. C. Singleton (Stanford Research Institute,

Sept, 1968). A discrete Fourier transform is defined by

$$Y(n) = \sum_{j=0}^{N-1} X(j)exp(\frac{-2\pi ijn}{N}), (n = 0, 1, \cdots, N-1);$$

and the inverse discrete Fourier transform is defined by

$$Z(n) = \sum_{j=0}^{N-1} X(j)exp(\frac{2\pi ijn}{N}), (n = 0, 1, \cdots, N-1);$$

satisfying the condition of $Z(j) = NX(j), (j = 0, 1, \cdots, N-1)$. Function **fft**() evaluates these sums using fast Fourier transform technique. It is not required that N be a power of 2. One-, two-, and three-dimensional transforms can be performed.

Example 1

This example illustrates the use of the FFT. There is a signal

$$f(t) = \begin{cases} 9e^{-9t} & t \geq 0 \\ 0 & t < 0 \end{cases}$$

Analytically, the Fourier transform of function $f(t)$ is given by

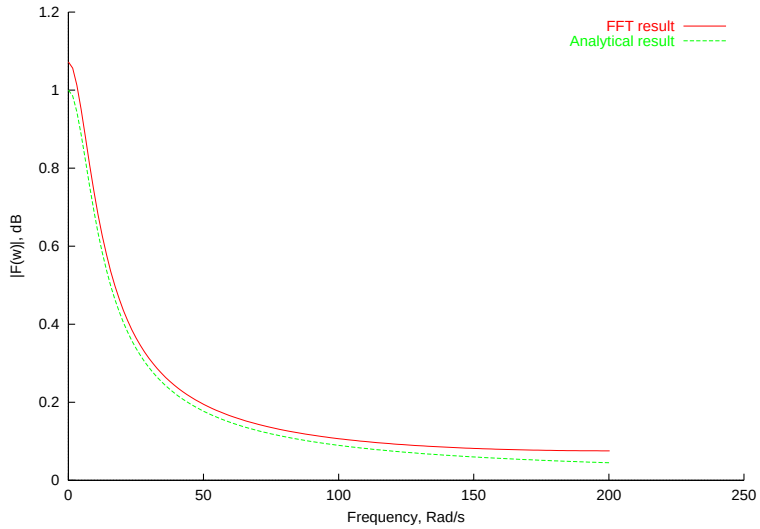
$$F(\omega) = \frac{9}{9 + j\omega}$$

This example illustrates an approach to estimate the Fourier transform of less common signals because the sample frequency is limited. In this example, total 256 points of signal are sampled in 4 seconds. So the sample rate is $256/4 = 64$Hz.

```
#include <stdio.h>
#include <math.h>
#include <chplot.h>
#include <numeric.h>

#define N 256    /* total sample points */
#define M 4      /* sample time, 4 seconds */

int main() {
   int i;
   array double t[N], f[N], E[N/2], W[N/2+1],y[2][N/2+1];
   array double complex F[N], Fp[N/2+1], Fa[N/2+1];
   double Ts, Ws;

   linspace(t, 0, M);
   f = 9*exp(-9*t);
   Ts = t[1] - t[0];
   Ws = 2*M_PI/Ts;
   fft(F,f);
   linspace(W,0,N/2);
   W = W*Ws/N;
   for (i=0; i<N/2+1; i++) {
      Fp[i] = F[i]*Ts;
      Fa[i] = 9.0/(9+W[i]*complex(0,1));
      y[0][i] = abs(Fp[i]);
```

```
      y[1][i] = abs(Fa[i]);
   }
   plotxy(W,y,"FFT example","Frequency, Rad/s","|F(w)|, dB");
}
```

Output



Example 2
This example shows the one-dimensional fft usage. When the optional argument *n* is specified, it takes *n* points FFT. If it is not specified, a total number of elements of *x* is taken to FFT by default.

```
#include <stdio.h>
#include <math.h>
#include <numeric.h>

#define N 16
#define M 4

int main() {
   int i,j,k,retval;
   array double x1[N];
   array double complex y1[N],y2[M],x2[N],x3[M];
   int n = M;

   linspace(x1, 0, 1);

   fft(y1,x1);
   ifft(x2,y1);
   printf("FFT number of points by default\n");
   printf("Original data    FFT result    FFT + iFFT result\n");
   for (i=0; i<N; i++)
   printf("%5.3f  %5.3f  %5.3f\n",x1[i],y1[i],x2[i]);

   printf("FFT number of points is specified by n=%d\n",n);
   fft(y2,x1,n);
   ifft(x3,y2,n);
   printf("\n\nOriginal data    FFT result    FFT + iFFT result\n");
   for (i=0; i<M; i++)
```

```
   printf("%5.3f  %5.3f  %5.3f\n",x1[i],y2[i],x3[i]);
}
```

Output

```
FFT number of points by default
Original data    FFT result    FFT + iFFT result
0.000  complex(8.000,0.000)  complex(-0.000,-0.000)
0.067  complex(-0.533,-2.681)  complex(0.067,0.000)
0.133  complex(-0.533,-1.288)  complex(0.133,-0.000)
0.200  complex(-0.533,-0.798)  complex(0.200,0.000)
0.267  complex(-0.533,-0.533)  complex(0.267,0.000)
0.333  complex(-0.533,-0.356)  complex(0.333,0.000)
0.400  complex(-0.533,-0.221)  complex(0.400,-0.000)
0.467  complex(-0.533,-0.106)  complex(0.467,0.000)
0.533  complex(-0.533,0.000)  complex(0.533,0.000)
0.600  complex(-0.533,0.106)  complex(0.600,0.000)
0.667  complex(-0.533,0.221)  complex(0.667,0.000)
0.733  complex(-0.533,0.356)  complex(0.733,0.000)
0.800  complex(-0.533,0.533)  complex(0.800,0.000)
0.867  complex(-0.533,0.798)  complex(0.867,0.000)
0.933  complex(-0.533,1.288)  complex(0.933,0.000)
1.000  complex(-0.533,2.681)  complex(1.000,-0.000)
FFT number of points is specified by n=4


Original data    FFT result    FFT + iFFT result
0.000  complex(0.400,0.000)  complex(0.000,0.000)
0.067  complex(-0.133,-0.133)  complex(0.067,0.000)
0.133  complex(-0.133,0.000)  complex(0.133,0.000)
0.200  complex(-0.133,0.133)  complex(0.200,0.000)
```

Example 3
This example shows the multi-dimensional fft usage. When the optional array argument *dim* is specified, the value for each element is used to specify the number of elements for each dimension of the array for performing FFT. If it is not specified, a total number of elements of *x* is taken to FFT by default.

```
#include <stdio.h>
#include <math.h>
#include <numeric.h>

#define N1 4
#define N2 3
#define M 3

int main() {
   int i,j,k,retval;
   array double complex x1[N1][N2],y1[N1][N2],x3[N1][N2];
   array double complex y[M][M],x2[M][M];
   int dim[2];

   linspace(x1, 0, 1);
   dim[0]=M;dim[1]=M;
   fft(y,x1,dim);
   dim[0]=M;dim[1]=M;
   ifft(x2,y,dim);

   printf("Input data =\n");
   printf("%5.3f",x1);
```

```
   printf("\nFFT result data =\n");
   printf("%5.3f",y);
   printf("\nFFT + iFFT result data =\n");
   printf("%5.3f",x2);

   fft(y1,x1);
   ifft(x3,y1);

   printf("\n\nInput data =\n");
   printf("%5.3f",x1);
   printf("\nFFT result data =\n");
   printf("%5.3f",y1);
   printf("\nFFT + iFFT result data =\n");
   printf("%5.3f",x3);
}
```

Output

```
Input data =
complex(0.000,0.000) complex(0.091,0.000) complex(0.182,0.000)
complex(0.273,0.000) complex(0.364,0.000) complex(0.455,0.000)
complex(0.545,0.000) complex(0.636,0.000) complex(0.727,0.000)
complex(0.818,0.000) complex(0.909,0.000) complex(1.000,0.000)

FFT result data =
complex(3.273,0.000) complex(-0.409,-0.236) complex(-0.409,0.236)
complex(-1.227,-0.709) complex(0.000,0.000) complex(0.000,0.000)
complex(-1.227,0.709) complex(0.000,-0.000) complex(0.000,-0.000)

FFT + iFFT result data =
complex(0.000,0.000) complex(0.091,0.000) complex(0.182,0.000)
complex(0.273,0.000) complex(0.364,0.000) complex(0.455,0.000)
complex(0.545,0.000) complex(0.636,0.000) complex(0.727,0.000)


Input data =
complex(0.000,0.000) complex(0.091,0.000) complex(0.182,0.000)
complex(0.273,0.000) complex(0.364,0.000) complex(0.455,0.000)
complex(0.545,0.000) complex(0.636,0.000) complex(0.727,0.000)
complex(0.818,0.000) complex(0.909,0.000) complex(1.000,0.000)

FFT result data =
complex(6.000,0.000) complex(-0.545,-0.315) complex(-0.545,0.315)
complex(-1.636,-1.636) complex(0.000,-0.000) complex(0.000,-0.000)
complex(-1.636,0.000) complex(-0.000,-0.000) complex(-0.000,0.000)
complex(-1.636,1.636) complex(0.000,0.000) complex(0.000,0.000)

FFT + iFFT result data =
complex(-0.000,0.000) complex(0.091,0.000) complex(0.182,0.000)
complex(0.273,0.000) complex(0.364,0.000) complex(0.455,0.000)
complex(0.545,0.000) complex(0.636,0.000) complex(0.727,0.000)
complex(0.818,0.000) complex(0.909,0.000) complex(1.000,0.000)
```

See Also

ifft().

References

R. C. Singleton, *An Algorithm for Computing the Mixed Radix F. F. T.*, IEEE Trans, Audio Electroacoust.,

AU-1(1969) 93-107.
MJ Oleson, *Netlib Repository at UTK and ORNL* , go/fft-olesen.tar.gz, http://www.netlib.org.

filter

Synopsis
#include <**numeric.h**>
int filter(**array double complex** *v*[&], **array double complex** *u*[&],
array double complex *x*[&], **array double complex** *y*[&], ...
/* [**array double complex** *zi*[&], [**array double complex** *zf*[&]] */);

Syntax
filter(*v*, *u*, *x*, *y*)
filter(*v*, *u*, *x*, *y*, *zi*, *zf*)

Purpose
Filters the data in vector *x* with a filter represented by vectors *u* and *v* to create the filtered data *y*. The filter is a direct form II transposed implementation of the standard difference equation:

$$\begin{aligned} y(n) &= v_0 * x(n) + v_1 * x(n-1) + ... + v_{nb} * x(n-nb-1) \\ &\quad -u_1 * y(n-1) - ... - u_{na} * y(n-na-1) \end{aligned}$$

The input-output description of this filtering operation in the z-transform domain is a rational transfer function

$$Y(z) = \frac{v_0 + v_1 z^{-1} + \cdots + v_{nb-1} z^{-nb-1}}{1 + u_1 z^{-1} + \cdots + u_{na-1} z^{-na-1}} X(z)$$

Return Value
This function returns 0 on success and -1 on failure.

Parameters
v A vector of size nb. It contains the numerator of polynomial coefficients of the filter.

u A vector of size na. It contains the denominator of polynomial coefficients of the filter.

x A vector of size N. It contains the raw data.

y A vector of size N. It contains the filtered data.

zi A vector of size $(\max(\text{na, nb}) - 1)$. It contains initial delays of the filter.

zf A vector of size max(na, nb). It contains the final delays of the filter.

Description
The numerator coefficients with zeros of the system transfer function *v*, denominator coefficients with poles of the system transfer function *u*, and vector *x* of input data can be of any supported arithmetic data type and size. Conversion of the data to double complex is performed internally. The vector *y* which is the same size

as *x* contains the result of filtered output. The optional arguments *zi* and *zf* are used to set the initial values of delays and get the final delays of the filter. They shall be double complex data type. The leading coefficient of denominator u_0 must be non-zero, as the other coefficients are divided by u_0.

Algorithm
The block diagram of the direct form II transposed implementation is shown in Figure 1.

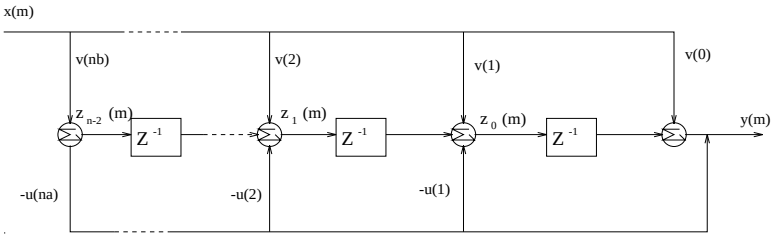


図 11.1: Direct form II transposed implementation

The operation of filter at sample m is given by the time domain difference equations

$$\begin{aligned} y(m) &= v_0 x(m) + z_0(m-1) \\ z_0(m) &= v_1 x(m) - u_1 y(m) + z_1(m-1) \\ \vdots &= \vdots \quad \vdots \\ z_{n-3}(m) &= v_{n-2} x(m) - u_{n-2} y(m) + z_{n-2}(m-1) \\ z_{n-2}(m) &= v_{n-1} x(m) - u_{n-1} y(m) \end{aligned}$$

where $n = \max(na, nb)$ and $m = 0, 1, \cdots, N$. Initial delay state $z_0(0), z_0(1), \cdots, z_0(n-2)$ are contained in vector *zi*. Final delay states $z_{(N-1)}(0), z_{(N-1)}(1), and z_{(N-1)}(n-1)$ are kept in vector *zf*.

Example 1
This example shows the usage of function of **filter**(), specification of the initial delays *zi* and the final delays *zf* are passed back to the calling function.

```
#include <stdio.h>
#include <numeric.h>

#define N1 6        /* x array size */
#define N2 1        /* B term coefficient number */
#define N3 4        /* A term coefficient number */

int main()
{
   int i,j,n,m;
   array double complex x[N1],v[N2],u[N3],zi[N3-1],zf[N3-1];
   array double x1[N1],v1[N2],u1[N3],y1[N1];
   array double complex y[N1];
   for (i=0;i<N1;i++){
      real(x[i])=i+1;
      imag(x[i])=i-1;
      x1[i] = i+1;
   }
   for (i=0;i<N3;i++) {
```

```
      real(u[i])=i+1;
      imag(u[i])=i+1;
      u1[i] = i+1;
   }
   for (i=0;i<N2;i++) {
      v[i]=i+1;
      v1[i] = i+1;
   }
   for (i=0;i<N3-1;i++) {
      zi[i]=1;
   }

   filter(v,u,x,y,zi,zf);  /* User specified initial delays */

   printf("u=%6.3f\n",u);
   printf("v=%6.3f\n",v);
   printf("x=%6.3f\n",x);
   printf("y=%6.3f\n",y);
   printf("zi=%6.3f\n",zi);
   printf("zf=%6.3f\n",zf);

   filter(v1,u1,x1,y1);        /* User did not specify initial delays */

   printf("u1=%6.3f\n",u1);
   printf("v1=%6.3f\n",v1);
   printf("x1=%6.3f\n",x1);
   printf("y1=%6.3f\n",y1);

   return 0;
}
```

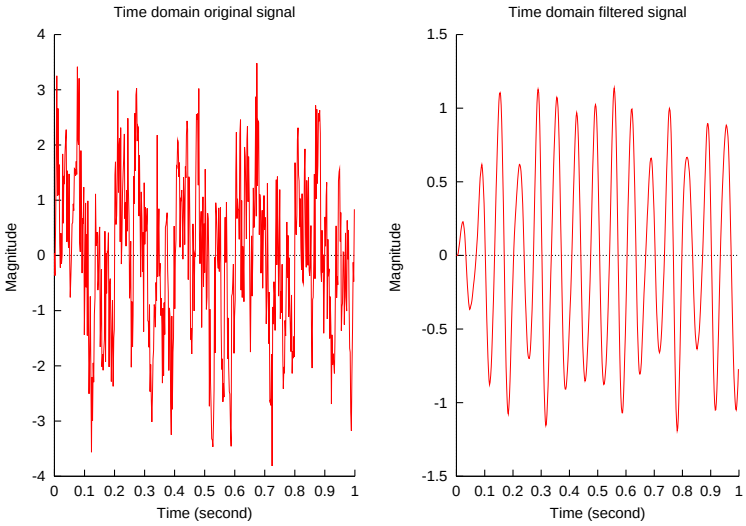
Output

```
u=complex( 1.000, 1.000) complex( 2.000, 2.000) complex( 3.000, 3.000)
complex( 4.000, 4.000)

v=complex( 1.000, 0.000)

x=complex( 1.000,-1.000) complex( 2.000, 0.000) complex( 3.000, 1.000)
complex( 4.000, 2.000) complex( 5.000, 3.000) complex( 6.000, 4.000)


y=complex( 1.000,-1.000) complex( 0.000, 1.000) complex( 0.000, 0.000)
complex(-1.000, 0.000) complex( 6.000,-5.000) complex(-4.000, 9.000)


zi=complex( 1.000, 0.000) complex( 1.000, 0.000) complex( 1.000, 0.000)


zf=complex(-6.000,-3.000) complex(-12.000,-7.000) complex(16.000,-36.000)


u1= 1.000  2.000  3.000  4.000

v1= 1.000

x1= 1.000  2.000  3.000  4.000  5.000  6.000

y1= 1.000  0.000  0.000  0.000  5.000 -4.000
```

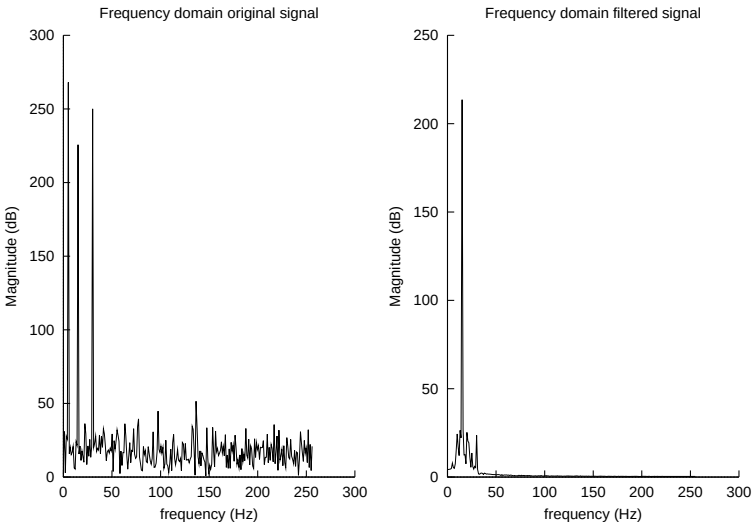
Example 2
This example shows how to use the FFT algorithm to find the spectrum of signals which is buried in noises and use the filter algorithm to filter unwanted signals.

```
#include <stdio.h>
#include <math.h>
#include <chplot.h>
#include <numeric.h>

#define N 512

int main() {
   array double t[N], x[N], y[N], Pyy[N/2], f[N/2], u[7], v[7];
   array double complex Y[N];
   int i;
   class CPlot plot;

   /* The filter arguments u[i] and v[i] are designed according to the characteristic
    of filter specified. In this example program, we use the designed filter to
   filter the raw data. */
   u[0]=1;u[1]=-5.66792131;u[2]=13.48109005;u[3]=-17.22250511;
   u[4]=12.46418230;u[5]=-4.84534157;u[6]=0.79051978;
   v[0]=0.00598202; v[1]=-0.02219918; v[2]=0.02645738; v[3]=0;
   v[4]=-0.02645738;v[5]=0.02219918;v[6]=-0.00598202;

   linspace(t,0,N-1);
   t = t/N;
   for (i=0; i<N; i++) {
      x[i] = sin(2*M_PI*5*t[i]) + sin(2*M_PI*15*t[i]) + sin(2*M_PI*t[i]*30);
      x[i]=x[i]+3*(urand(NULL)-0.5);
   }

   filter(v,u,x,y);
   plotxy(t,x,"Time domain original signal","Time (second)","Magnitude ");
   plotxy(t,y,"Time domain filtered signal","Time (second)","Magnitude ");

   fft(Y,x);
   for (i=0; i<N/2; i++)
      Pyy[i] = abs(Y[i]);
   linspace(f,0,N/2);
   plotxy(f,Pyy,"Frequency domain original signal","frequency (Hz)","Magnitude (db)");
   fft(Y,y);
   for (i=0; i<256; i++)
      Pyy[i] = abs(Y[i]);
   linspace(f,0,255);
   plotxy(f,Pyy,"Frequency domain filtered signal","frequency (Hz)","Magnitude (db)");
}
```

Output
This figure shows singals in the time domain. The figure on the left hand side is the original signals with three sinusoidal components at frequency of 5, 15, 30 Hz which are buried in white noises. The figure on the right hand side is signals after a sixth order IIR filter with a passband of 10 to 20 Hz. We hope to keep the 15 Hz sinusoidal signals and get rid of the 5 and 30 Hz sinusoids and other white noise using a filter with coefficients *u* and *v* given in the program.



The signals in the frequency domain are obtained by a fast Fourier transform algorthm using function **fft**(). This figure shows the original and filtered signals in the frequency domain. The figure on the left hand side shows the original signals with three separate main frequency spectrums and some noise frequency. After filtering, the signals, as shown on the right hand side of the figure , contain mainly a frequency of 15 Hz and a few noise components.



See Also

filter2().

References

Alon V. Oppenheim, *Discrete-time Signal Processing*, Prentice Hall, 1998.

filter2

Synopsis
#include <**numeric.h**>
int filter2(**array double complex** *y*[&][&], **array double complex** *u*[&][&], **array double complex** *x*[&][&],
... /* [**string_t** *method*] */);

Syntax
filter2(*y*, *u*, *x*)
filter2(*y*, *u*, *x*, *method*);

Purpose
Two-dimensional Discrete Fourier Transform based FIR filter.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
y Two-dimensional **array** of filtered data.

u Two-dimensional **array** of size $nu \times mu$ of FIR filter.

x Two-dimensional **array** of size $nx \times mx$ of raw data.

method A string which describes the filter output method.

Description
Filter the data *x* with the 2-D FIR filter in the matrix *u*. By default, the result *y* is computed using 2-D convolution with the same size as *x*. If option *method* is specified, *y* is computed via 2-D convolution with size specified by ”method”:
”same” - (default) returns the central part of the convolution that is the same size as *x*.
”valid” - returns only those parts of the convolution that are computed without the zero-padded edges, the size of *y* is $(nx - nu + 1) \times (mx - mu + 1)$. The size of *x* must be bigger than the size of *u*.
”full” - returns the full 2-D convolution, that is the size of *y* is $(nx + nu - 1) \times (mx + mu - 1)$.

Algorithm
This function calls **conv2**(), the two-dimensional convolution function, to implement the filtering operation. **See conv2**().

Example
In image processing, Sobel filter often used to smoothing image. Convoluting the original two-dimensional

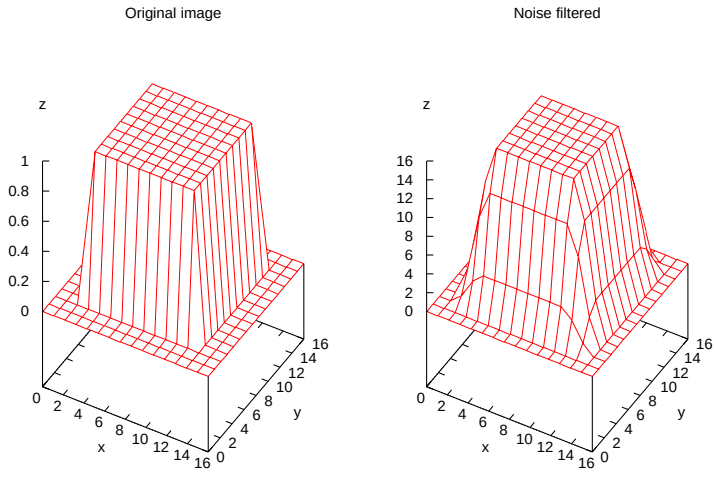
image data with Sobel mask

$$s = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

The image can be smoothed.

```
#include <math.h>
#include <chplot.h>
#include <numeric.h>

int main() {
   int i, j;
   array double u[3][3]={{1,2,1},{2,4,2},{1,2,1}};
   array double x[16],y[16],z1[256],z[16][16],Z[18][18],Z1[256];

   linspace(x,0,16);
   linspace(y,0,16);
   for(i=3; i<13; i++)
      for(j=3; j<13; j++) {
         z1[i*16+j]=1;
         z[i][j] = 1;
   }
   plotxyz(x,y,z1);
   filter2(Z,u,z,"full");
   for(i = 0; i<16; i++)
      for(j=0; j<16; j++)
         Z1[i*16+j] = Z[i+1][j+1];
   plotxyz(x,y,Z1);
}
```

Output



See Also

filter(), **conv2**().

findvalue()

Synopsis
#include <**numeric.h**>
int findvalue(**array int** *y[&]*, **array double complex** &*x*);

Syntax
findvalue(*y*, *x*)

Purpose
Obtain indices of nonzero elements of an array.

Return Value
This function returns the number of nonzero elements of an array.

Parameters
y A vector the size of the total number of elements of array *x*. It contains the indices of nonzero elements of array *x*.

x Any-dimensional input array which contains the original data.

Description
The original data **array** *x* can be of any supported arithmetic data type and extent sizes. The total number of nonzero elements is returned by the function call of **findvalue**(). Vector *y* is of integer data type with the total number of elements of *x*. It contains the indices of nonzero elements of array *x*. The remaining elements of *y* contain a value of -1. If *x* is an array of complex data type, a zero element has a value of zero for both its real and imaginary parts.

Example

```
#include <numeric.h>
int main() {
   array double x1[5]={0,43.4,-7,-2.478,7};
   array double x[4][2]={1,3,
                         2.45,8.56,
                         -3,5,
                         6,8};
   array double complex zx[3][2]={complex(1,1),0,
                         3,complex(4,1),
                         complex(1,0),complex(0,1)};
   array int zy[6], y1[5],y[8];
   int i,j;

   j=findvalue(y1,x1);
   printf("x1 = \n%f",x1);
   printf("Total %d of x1 are not eq. 0. They are located at y1 = \n",j);
   for (i=0; i<j; i++) printf("%d   ",y1[i]);
```

```
   printf("\n");
   j=findvalue(y,x<2.0);
   printf("x = \n%f",x);
   printf("Total %d of x are less than 2.0. They are located at y = \n",j);
   for (i=0; i<j; i++) printf("%d   ",y[i]);
   printf("\n");
   j=findvalue(zy,zx);
   printf("zx = \n%f",zx);
   printf("Total %d of zx are not eq. 0. They are located at zy = \n",j);
   for (i=0; i<j; i++) printf("%d   ",zy[i]);
   printf("\n");
}
```

Output

```
x1 =
0.000000 43.400000 -7.000000 -2.478000 7.000000
Total 4 of x1 are not eq. 0. They are located at y1 =
1   2   3   4
x =
1.000000 3.000000
2.450000 8.560000
-3.000000 5.000000
6.000000 8.000000
Total 2 of x are less than 2.0. They are located at y =
0   4
zx =
complex(1.000000,1.000000) complex(0.000000,0.000000)
complex(3.000000,0.000000) complex(4.000000,1.000000)
complex(1.000000,0.000000) complex(0.000000,1.000000)
Total 5 of zx are not eq. 0. They are located at zy =
0   2   3   4   5
```

fliplr()

Synopsis
#include <**numeric.h**>
int fliplr(**array double complex** *y[&][&]*, **array double complex** *x[&][&]*);

Syntax
fliplr(*y*, *x*)

Purpose
Flip matrix in left/right direction.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

x Two-dimensional matrix which contains the original data.

y Two-dimensional matrix of the same data type and size as *x*, which contains the flipped result of input matrix *x*.

Description
This function flips matrix *x* in the left/right direction with rows being preserved and columns flipped.

Example

```
#include <numeric.h>
int main() {
   array double x[2][4]={1,3, 2.45,8.56,
                         3,5, 6,8};
   array double complex zx[2][3]={complex(1,1),0,3,
                         complex(4,1), 6,0};
   array double complex zy[2][3];
   array double y[2][4];

   fliplr(y,x);
   printf("x = \n%f",x);
   printf("y = \n%f",y);
   printf("\n");

   fliplr(zy,zx);
   printf("zx = \n%5.1f",zx);
   printf("zy = \n%5.1f",zy);
}
```

Output

```
x =
1.000000 3.000000 2.450000 8.560000
3.000000 5.000000 6.000000 8.000000
y =
8.560000 2.450000 3.000000 1.000000
8.000000 6.000000 5.000000 3.000000

zx =
complex(  1.0,  1.0) complex(  0.0,  0.0) complex(  3.0,  0.0)
complex(  4.0,  1.0) complex(  6.0,  0.0) complex(  0.0,  0.0)
zy =
complex(  3.0,  0.0) complex(  0.0,  0.0) complex(  1.0,  1.0)
complex(  0.0,  0.0) complex(  6.0,  0.0) complex(  4.0,  1.0)
```

See Also

flipud(), **rot90**().

flipud()

Synopsis
#include <**numeric.h**>
int flipud(array double complex *y[&][&]*, **array double complex** *x[&][&]*);

Syntax
flipud(*y*, *x*)

Purpose
Flip matrix in up/down direction.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Two-dimensional matrix which contains the original data.

y Two-dimensional matrix of the same data type and size as *x*, which contains the flipped result of input matrix *x*.

Description
This function flips matrix *x* in the up/down direction with columns being preserved and rows flipped.

Example

```
#include <numeric.h>
int main() {
   array double x[4][2]={1,3,
                         2.45,8.56,
                         3,5,
                         6,8};
   array double complex zx[3][2]={complex(1,1),0,
                         3,complex(4,1),
                         6,0};
   array double complex zy[3][2];
   array double y[4][2];

   flipud(y,x);
   printf("x = \n%f",x);
   printf("y = \n%f",y);
   printf("\n");

   flipud(zy,zx);
   printf("zx = \n%5.1f",zx);
   printf("zy = \n%5.1f",zy);
}
```

Output

```
x =
1.000000 3.000000
2.450000 8.560000
3.000000 5.000000
6.000000 8.000000
y =
6.000000 8.000000
3.000000 5.000000
2.450000 8.560000
1.000000 3.000000

zx =
complex(  1.0,  1.0) complex(  0.0,  0.0)
complex(  3.0,  0.0) complex(  4.0,  1.0)
complex(  6.0,  0.0) complex(  0.0,  0.0)
zy =
complex(  6.0,  0.0) complex(  0.0,  0.0)
complex(  3.0,  0.0) complex(  4.0,  1.0)
complex(  1.0,  1.0) complex(  0.0,  0.0)
```

See Also

fliplr(), **rot90**().

fminimum

Synopsis
#include <**numeric.h**>
int fminimum (**double** **fminval*, **double** **xmin*, **double** (**func*)(**double**), **double** *x0*, **double** *xf*, ...
/* [**double** *rel_tol*, **double** *abs_tol*] */);

Syntax
fminimum(*fminval, xmin, func, x0, xf*)
fminimum(*fminval, xmin, func, x0, xf, rel_tol*)
fminimum(*fminval, xmin, func, x0, xf, rel_tolm, abs_tol*)

Purpose
Find the minimum value of a one-dimensional function and its corresponding position for the minimum value.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
fminval A pointer to the variable which passes the calculated minimum value of the function.

xmin A pointer to the variable which contains the calculated position with the minimum value of the function.

func The function given by user for minimization.

x0 The beginning-point of the interval on which a minimum value is searched for.

xf The end-point of the interval on which a minimum value is searched for.

rel_tol The relative tolerance of the calculation.

abs_tol The absolute tolerance of the calculation.

Description
Determines a point between *x0* and *xf* at which the real function *func* assumes a minimum value. The function *func* given by the user has an argument for the *x* value input. The argument *fminval* passes the calculated minimum value of the function. The argument *xmin* contains the position where the minimum value of the function is found. The tolerance is defined as a function of x: $|x|rel\_tol + abs\_tol$, where *rel_tol* is the relative precision and *abs_tol* is the absolute precision which should not be chosen equal to zero. The default value for *rel_tol* and *abs_tol* is 10^{-6}.

Limitations
In the interval at which the function assumes a minimum value, it is assumed that for some point *u* either (a)

func is strictly monotonically decreasing on [a,u) and strictly monotonically increasing on [u,b] or (b) these two intervals may be replaced by [a,u] and (u,b] respectively.

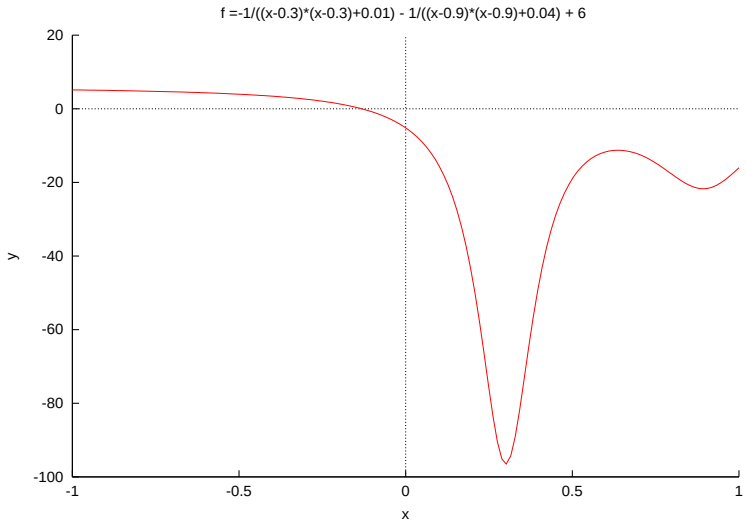
Example

Find the minimum of the function $f(x) = -\frac{1}{(x-0.3)^2+0.01} - \frac{1}{(x-0.9)^2+0.04} - 6$ at intervals [-1,0], [0,0.8], [0.4,1] and [0,1] with the default value for tolerance. For interval [0,1] this function can only find the local minimum at 0.892716. This example also finds the minimum of the function at the interval [0,0.8] with the given tolerance rel_tol and abs_tol equal to 0.01.

```
#include <numeric.h>
#include <stdio.h>

double func(double x) {
    double y;
    y = 1/((x-0.3)*(x-0.3)+0.01) + 1/((x-0.9)*(x-0.9)+0.04) - 6;
    return -y;
}

int main() {
    double x0, xf, rel_tol, abs_tol;
    double func(double);
    double xmin, fminval;
    int status;
    string_t title = "f =-1/((x-0.3)*(x-0.3)+0.01) - 1/((x-0.9)*(x-0.9)+0.04) + 6 ";

    fplotxy(func, -1, 1, 50, title, "x", "y");
    x0 = -1;  xf = 0; /* fminimum is at 0.0 */
    status = fminimum(&fminval, &xmin, func, x0, xf);
    printf("Interval = [%3.2f, %3.2f]\n", x0, xf);
    printf("status = %d\n", status);
    printf("xmin = %f\n", xmin);
    printf("m = %f\n\n", fminval);

    x0 = 0;  xf = 0.8;  /* fminimum is around 0.3 */
    status = fminimum(&fminval, &xmin, func, x0, xf);
    printf("Interval = [%3.2f, %3.2f]\n", x0, xf);
    printf("status = %d\n", status);
    printf("xmin = %f\n", xmin);
    printf("m = %f\n\n", fminval);

    rel_tol = 1E-2; /* default tolerance is 1E-6 */
    abs_tol = 1E-2; /* default tolerance is 1E-6 */
    x0 = 0;  xf = 0.8;  /* fminimum is around 0.6 */
    status = fminimum(&fminval, &xmin, func, x0, xf,rel_tol,abs_tol);
    printf("Interval = [%3.2f, %3.2f]\n", x0, xf);
    printf("status = %d\n", status);
    printf("xmin = %f\n", xmin);
    printf("m = %f\n\n", fminval);

    x0 = 0.4;  xf = 1; /* fminimum is at 0.4 */
    status = fminimum(&fminval, &xmin, func, x0, xf);
    printf("Interval = [%3.2f, %3.2f]\n", x0, xf);
    printf("status = %d\n", status);
    printf("xmin = %f\n", xmin);
    printf("m = %f\n\n", fminval);
```

```
    x0 = 0;  xf = 1; /* Only local minimum at 0.892716 is found */
    status = fminimum(&fminval, &xmin, func, x0, xf);
    printf("Interval = [%3.2f, %3.2f]\n", x0, xf);
    printf("status = %d\n", status);
    printf("xmin = %f\n", xmin);
    printf("m = %f\n\n", fminval);
}
```

Output




```
Interval = [-1.00, 0.00]
status = 0
xmin = 0.000000
m = -5.176471

Interval = [0.00, 0.80]
status = 0
xmin = 0.300375
m = -96.501409

Interval = [0.00, 0.80]
status = 0
xmin = 0.305573
m = -96.232705

Interval = [0.40, 1.00]
status = 0
xmin = 0.400000
m = -47.448276

Interval = [0.00, 1.00]
status = 0
xmin = 0.892717
m = -21.734573
```

See Also

fminimums().

References

H. T. Lau, *A Numerical Library in C for Scientists and Engineers*, CRC Press, Inc. 1995.

fminimums

Synopsis
#include <**numeric.h**>
int fminimums(**double** **fminval*, **double** *xmin*[&], **double** (**func*)(**int**, **double**[ ]), **double** *x0*[&], ...
/*[**double** *rel_tol*, **double** *abs_tol*, **int** *numfuneval*] */);

Syntax
fminimums(*fminimumval, xmin, func, x0)*
fminimums(*fminimumval, xmin, func, x0, rel_tol)*
fminimums(*fminimumval, xmin, func, x0, rel_tol, abs_tol)*
fminimums(*fminimumval, xmin, func, x0, rel_tol, abs_tol, numfuneval)*

Purpose
Find the minimum position and value of an n-dimensional function.

Return Value
This function returns one of the following values: **Parameters**
0 On successful.

-1 The process is broken off because at the end of an iteration step the number of calls of *func* exceeded the value of *numfuneval*.

-2 The process is broken off because the condition of the problem is too bad.

-3 Function not supported.

Parameters
fminval A pointer to the variable which contains the calculated minimum value of the function.

xmin A pointer to the array which contains the calculated positions with the minimum value of the function.

func An n-dimensional function given by the user for minimization.

x0 An input array of approximation for variables *x* where the function gives the minimum value.

rel_tol The relative tolerance of the calculation.

abs_tol The absolute tolerance of the calculation.

numfunceval The maximum number of function evaluation allowed.

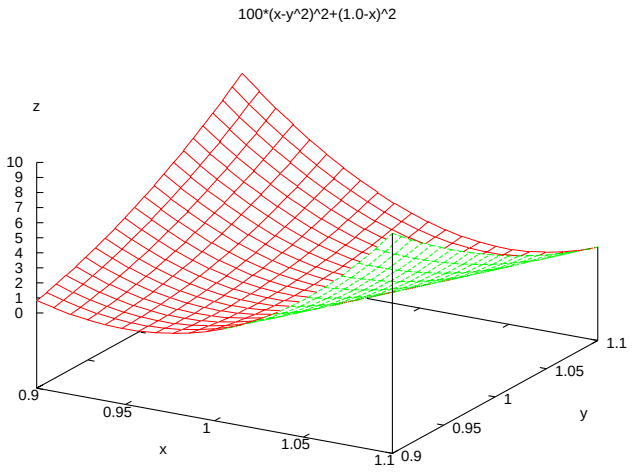
Description
Given an n-dimensional function and an array which contains initial estimate by the user as input, this function calculates the array of the position at which the function has a minimum value. The number of dimension is taken from input array *x* internally. The function *func* should deliver the value of the function

to be minimized, at the points given by *x*. The tolerance is defined as a function of *x* as $|x|rel\_tol + abs\_tol$, where *rel_tol* is the relative precision and *abs_tol* is the absolute precision which should not be chosen equal to zero. The argument *numfunceval* is the maximum number of function evaluations allowed. The default values for *rel_tol, abs_tol* and *numfuneval* are $10^{-6}, 10^{-6}$, and 250, respectively.

Example
Find the minimum value of two dimension function $100(x_1 - x_0^2)^2 + (1.0 - x_0)^2$ with the initial guess of $x_0 = -1.2$ and $x_1 = 1.0$.

```
#include <numeric.h>
#include <stdio.h>
#define N 2

double func(double x[]) {
    double temp;
    temp=x[1]-x[0]*x[0];
    return 100*temp*temp+(1.0-x[0])*(1.0-x[0]);
}

int main () {
    double func(double x[]);
    double fminval,xmin[N],x0[N],rel_tol, abs_tol;
    int numfuneval, status;
    x0[0] = -1.2;
    x0[1]=1.0;

    /* Plot a 2 dimensional graph of the function
       for which the minimum will be found */
    double funcp(double x, y)
    {
        double temp;
        temp=y-x*x;
        return 100*temp*temp+(1.0-x)*(1.0-x);
    }
    fplotxyz(funcp, 0.9, 1.1, 0.9, 1.1, 22,22,"100*(x-y*y)*(x-y*y)+(1.0-x)*(1.0-x)",
             "x","y","z");

    status = fminimums(&fminval, xmin, func,x0);
    printf("status = %d\n", status);
    printf("fminval = %f\n", fminval);
    printf("xmin[0]  = %f\n", xmin[0]);
    printf("xmin[1]  = %f\n", xmin[1]);

    rel_tol = 1.0e-3;   /* not default value */
    abs_tol = 1.0e-3;   /* not default value */
    numfuneval = 250;   /* not default value */
    status = fminimums(&fminval, xmin, func,x0,rel_tol, abs_tol,numfuneval);
    printf("status = %d\n", status);
    printf("fminval = %f\n", fminval);
    printf("xmin[0]  = %f\n", xmin[0]);
    printf("xmin[1]  = %f\n", xmin[1]);
}
```

Output




```
status = 0
fminval = 0.000000
xmin[0]  = 1.000000
xmin[1]  = 1.000000
status = 0
fminval = 0.000000
xmin[0]  = 0.999999
xmin[1]  = 1.000028
```

See Also

fminimum().

References

H. T. Lau, *A Numerical Library in C for Scientists and Engineers*, CRC Press, Inc. 1995.

fsolve

Synopsis
#include <**numeric.h**>
int fsolve (**double** *x*[:], **void**(**func*)(**double** *x*[ ], **double** *y*[ ]), **double** *x0*[:]);

Purpose
Find a zero position of a nonlinear system of equations.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x An n-dimensional array which contains the calculated zero position.

x0 An n-dimensional array which contains the initial guess for a zero position.

func A pointer to the function given by the user.

Description
Function **fsolve**() finds a zero position of function *func*(**x**) provided by the user. The user function has two arguments, the first one for input and the second one for output. The input argument is an n-dimensional array, and the function values calculated will be delivered by the second array argument of the same dimension as output. The number of dimension is taken from the function given by the user internally.

Algorithm
The multidimensional nonlinear system of equations is solved by a multidimensional secant method: Broyden's method. Since the Jacobian matrix which is needed for solution of the equations is approximated by Quisi-Newton method inside the function, the user does not need to provide analytic formula for derivatives of equations. Broyden's method converges superlinearly once you get close enough to the root.

Example
Solve the following nonlinear system of two equations

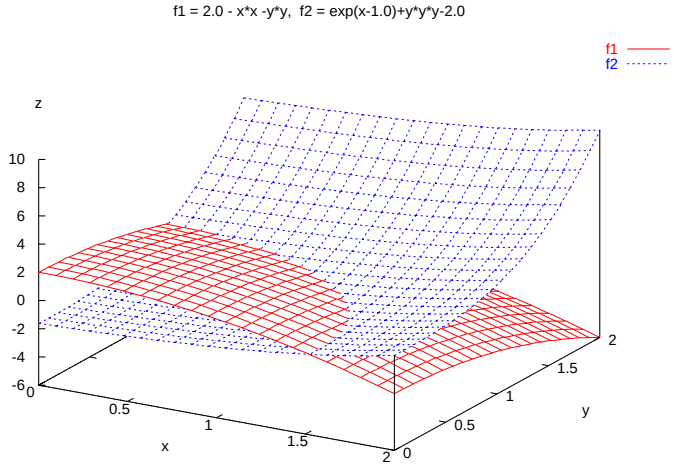
$$\begin{aligned} f_0 &= -(x_0^2 + x_1^2 - 2.0) = 0 \\ f_1 &= e^{x_0 - 1.0} + x_1^3 - 2.0 = 0 \end{aligned}$$

with the initial guesses of zero point at $x_0 = 2.0$, and $x_1 = 0.5$.

```
/* Driver for routine fsolve: for 2 dimentional equation*/
#include <stdio.h>
#include <numeric.h>
#include <chplot.h>
#define N 2
#define NP 22
```

```
void func(double x[], double f[]){
    f[0]=-(x[0]*x[0]+x[1]*x[1]-2.0);
    f[1]=exp(x[0]-1.0)+x[1]*x[1]*x[1]-2.0;
}

int main(){
    double x[N], x0[N], f[N];
    double x1[NP], x2[NP], f1[NP*NP], f2[NP*NP];
    int i,j,status, datasetnum=0;
    class CPlot plot;

    linspace(x1, 0, 2);
    linspace(x2, 0, 2);

    for(i=0; i<NP; i++) {
        for(j=0; j<NP; j++){
            f1[NP*i+j] = -(x1[i]*x1[i]+x2[j]*x2[j] - 2.0);
            f2[NP*i+j] = exp(x1[i]-1.0) + x2[j]*x2[j]*x2[j]-2.0;
        }
    }
    plot.data3D(x1, x2, f1);
    plot.data3D(x1, x2, f2);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.legend("f1", datasetnum);
    datasetnum++;
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.legend("f2", datasetnum);
    plot.label(PLOT_AXIS_X, "x");
    plot.label(PLOT_AXIS_Y, "y");
    plot.label(PLOT_AXIS_Z, "z");
    plot.title("f1 = 2.0 - x*x -y*y,  f2 = exp(x-1.0)+y*y*y-2.0");
    plot.ticsLevel(0);
    plot.colorBox(PLOT_OFF);
    plot.plotting();

    x0[0]=2.0;
    x0[1]=0.5;
    status=fsolve(x, func, x0);

    func(x,f);

    printf("status = %d\n",status);
    if (status==-1)
        fprintf(stderr, "Convergence problems.\n");
    printf("%7s %3s %12s\n","Index","x","f");
    for (i=0;i<2;i++)
        printf("%5d %12.6f %12.6f\n",i,x[i],f[i]);
}
```

Output




```
status = 0
  Index    x             f
    0      1.000016     -0.000020
    1      0.999994     -0.000002
```

See Also

fzero().

References

William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery, *Numeric Recipes in C*, Second Edition, Cambridge University Press, 1997.

funm

Synopsis
#include <**numeric.h**>
int funm(**array double** *y[&][&]*, **double** (**func*)(**double**), **array double** *x[&][&]*);

Syntax
funm(*y*, *func*, *x*)

Purpose
Evaluate general real matrix function.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Input square matrix. It contains data to be evaluated.

func A function routine given by the user.

y Output square matrix which contains data of the calculated function values.

Description
This function evaluates the matrix version of the function specified by argument *func*. In this function, The input matrix *x* shall be **double** data type and the specified function prototype shall be **double** *func*(**double**). The output matrix *y* could be **real** or **complex** data type as required.

Example
A real matrix evaluation.

```
#include <numeric.h>
double mylog(double x) {
   return log(x);
}

int main() {
   array double x[3][3]={1,2,3,
                         3,4,5,
                         6,7,8};
   array double y[3][3];

   expm(y,x);
   printf("x = \n%f",x);
   printf("y = \n%f",y);
   funm(x,mylog,y);
   printf("x = \n%f",x);

}
```

Output

```
x =
1.000000 2.000000 3.000000
3.000000 4.000000 5.000000
6.000000 7.000000 8.000000
y =
157372.953093 200034.605129 242697.257164
290995.910241 369883.552084 448769.193928
491431.845963 624654.472518 757878.099072
x =
1.000000 2.000000 3.000000
3.000000 4.000000 5.000000
6.000000 7.000000 8.000000
```

See Also

expm(), **logm**(), **cfunm**(), **sqrtm**().

References

G. H. Golub, C. F. Van Loan, Matrix Computations Third edition, The Johns Hopkins University Press, 1996

fzero

Synopsis

#include <**numeric.h**>

int fzero(**double** **x*, **double**(**func*)(**double** *x*), ... /* [**double** *x0*] | [**double** *x02*[2]*/);

Purpose

Find a zero position of a nonlinear function with one variable.

Return Value

This function returns 0 on success and -1 on failure.

Parameters

x Output of the calculated zero position.

func A pointer to the function given by the user.

x0 Input of the initial guess for zero position.

x02 Input of a vector of length 2 and double type. The function shall be bracketed in the interval of [*x02*[0], *x02*[1]] so that the sign of *func*(*x02*[0]) differs from the sign of *func*(*x02*[1]). Otherwise, an error occurs.

Description

Function **fzero**() finds a zero position of function *func*(*x*) provided by the user. The input argument of function *func*() is *x* value.

Algorithm

If x0 is given, the algorithm of one-dimensional zero finding is based on the algorithm for finding zero of the multi-dimensional nonlinear system of equations. Based on **fsolve**(), the number of dimension is set to 1 for **fzero**(). See algorithm for **fsolve**(). If x02 is given, a bisection method is used to find the zero position.

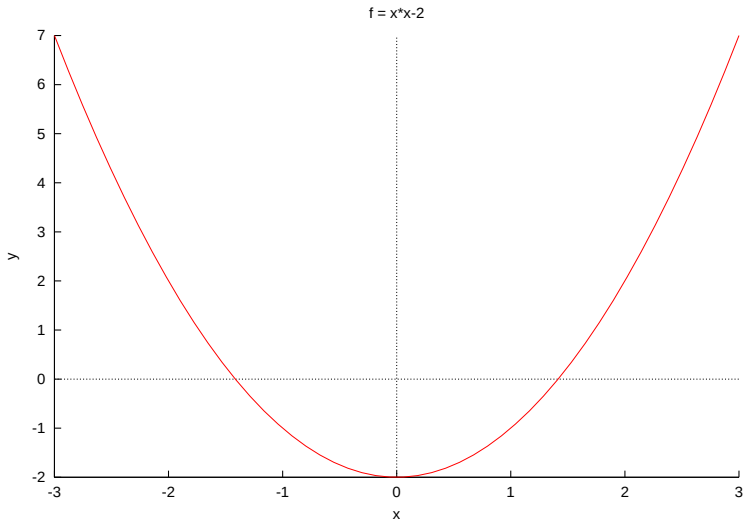
Example

Find zero of function $f = x^2 - 2$ with initial guess $x = 2.0$.

```
#include <stdio.h>
#include <chplot.h>
#include <numeric.h>

double func(double x) {
    return x*x-2.0;
}

int main() {
    double x, x0, f, x02[2];
    int i,status;
    double func(double);
    fplotxy(func, -3, 3, 50, "f = x*x-2", "x", "y");

    x0=-2.0;
```

```
    status=fzero(&x, func, x0); // x0 is a scalar
    f=func(x);
    if (status<0)
        printf("fzero() failed.\n");
    printf(" %6s %12s\n","x","f");
    printf("%10.6f %12.6f\n",x,f);

    x02[0]=-2.0;
    x02[1]=0;
    status=fzero(&x, func, x02); // x02 is an array
    f=func(x);
    if (status<0)
        printf("fzero() failed.\n");
    printf(" %6s %12s\n","x","f");
    printf("%10.6f %12.6f\n",x,f);
}
```

Output




```
     x            f
 1.414213    -0.000000
     x            f
-1.414213    -0.000000
```

See Also

fsolve().

References

William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery, *Numeric Recipes in C*, Second Edition, Cambridge University Press, 1997.

gcd()

Synopsis
#include <**numeric.h**>
int gcd(array int &*u*, **array int** &*v*, **array int** &*g*, ... /* [**array int** *c[&]*, **array int** *d[&]*]*/);

Syntax
gcd(*u*, *v*, *g*)
gcd(*u*, *v*, *g*, *c*, *d*)

Purpose
Obtain the greatest common divisor of the corresponding elements of two arrays of integer type.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
u Any-dimensional array which contains non-negative integer elements for the greatest common divisor calculation.

v An array the same dimension and size as *u*. It contains another non-negative integer elements for the greatest common divisor calculation.

g An output array the same dimension and size as *u*. It contains the result of the greatest common divisor calculation of *u* and *v*.

c An output array the same dimension and size as *u*.

d An output array the same dimension and size as *u*.

Description
This function calculates the greatest common divisor of corresponding elements *u* and *v*. The arrays *u* and *v* shall be the same size and contain non-negative integer data. The output array *g* is a positive integer array of the same size as *u*. The optional output arrays *c* and *d* are of the same size as *u* and satisfy the equation $u. * c + v. * d = g$.

Example

```
#include <numeric.h>
int main() {
   array int u[2][3] = {1,2,7,15,3,4};
   array int v[2][3] = {2,4,8,3,8,3};
   array int g[2][3],c[2][3],d[2][3];
   gcd(u,v,g,c,d);
   printf("g=%d",g);
```

```
   printf("c=%d",c);
   printf("d=%d",d);
   printf("u.*c+v.*d = %d",u.*c+v.*d);
}
```

Output

```
g=1 2 1
3 1 1
c=1 1 -1
0 3 1
d=0 0 1
1 -1 -1
u.*c+v.*d = 1 2 1
3 1 1
```

See Also

lcm().

References

Knuth, Donald Ervin, *The art of computer programming*, Vol. 2, Addison-Wesley Pub. Co., 1973.

getnum

Synopsis
#include <**numeric.h**>
double getnum(string_t *msg*, **double** *d*);

Purpose
Obtain a number from the console through standard input stream *stdin*.

Return Value
This function returns the default number or input number from the console.

Parameters
msg Message printed out in the standard output stream *stdout*.

d Default number.

Description
This function returns the default number when carriage RETURN is entered as input or new number from *stdin* as a double-precision floating-point number. However, if an invalid number is entered, a number is requested. The message in string *msg* will be printed out.

Example

```
#include <numeric.h>
/* The input numbers typed in on the console are 100 200 */

int main() {
    double d=2.0;

    printf("\nEnter a number [%lf]: ",d);
    d = getnum(NULL, d);
    printf("\nThe number you entered: %lf\n\n",d);
    d = getnum("Please enter a number[10.0]: ", 10);
    printf("\nThe number you entered: %lf\n\n",d);
}
```

Output

```
Enter a number [2.000000]:
The number you entered: 100.000000

Please enter a number[10.0]:

The number you entered: 200.000000
```

See Also
getline().

References

hessdecomp

Synopsis
#include <**numeric.h**>
int hessdecomp(**array double complex** *a*[&][&], **array double complex** *h*[&][&], ...
/* [**array double complex** *p*[&][&]] */);

Syntax
hessdecomp(*a*, *h*);
hessdecomp(*a*, *h*, *p*);

Purpose
Reduces a real general matrix *a* to upper Hessenberg form *h* by an orthogonal/unitary matrix *p* similarity transformation: $p^T * a * p = h$ or $p^H * a * p = h$ for real and complex, respectively. The p^H is Hermitian of matrix *p*.

Return Value
This function returns 0 on success and negative value on failure.

Parameters
a A $n \times n$ square matrix to be decomposed.

h An output two-dimensional matrix which contains the upper Hessenberg matrix of matrix *a*.

p An optional output two-dimensional array which contains an orthogonal or unitary matrix.

Description
This function computes the Hessenberg matrix *h* and an orthogonal/unitary matrix *p* so that $h = p^T * a * p$ and $p^T * p = I$ for real matrix, and $h = p^H * a * p$ and $p^H * p = I$ for complex matrix. Each element of a Hessenberg matrix below the first subdiagonal is zero. If the matrix is symmetric or Hermitian, the form is tridiagonal. This matrix has the same eigenvalues as the original one, but less computation is needed to calculate them.
In this function, square matrix *a* could be any supported arithmetic data type. Output *h* is the same dimension and data type as input *a*. If the input *a* is of real type, the optional output *p* shall only be double type. If the input *a* is complex type, *p* shall be **complex** or **double complex** type.

Example1
Reduce a real general matrix to upper Hessenberg form.

```
#include <numeric.h>
int main() {
     int m = 3;
     array double a[3][3] = { 0.8,0.2,0.1,
                              0.1,0.7,0.3,
                              0.1,0.1,0.6};
```

```
     int status;
     array double p[m][m], h[m][m];
     status = hessdecomp(a, h);
     if (status == 0) {
        printf("h =\n%f\n", h);
     }
     else
        printf(" Hessenberg matrix calculation error.\n");
     status = hessdecomp(a, h, p);
     if (status == 0) {
        printf("h =\n%f\n", h);
        printf("p =\n%f\n", p);
        printf("transpose(p)*p =\n%f\n",transpose(p)*p );
        printf("transpose(p)*a*p - h =\n%f\n", transpose(p)*a*p - h);
     }
     else
        printf(" Hessenberg matrix calculation error.\n");
}
```

Output1

```
h =
0.800000 -0.212132 -0.070711
-0.141421 0.850000 -0.050000
0.000000 0.150000 0.450000

h =
0.800000 -0.212132 -0.070711
-0.141421 0.850000 -0.050000
0.000000 0.150000 0.450000

p =
1.000000 0.000000 0.000000
0.000000 -0.707107 -0.707107
0.000000 -0.707107 0.707107

transpose(p)*p =
1.000000 0.000000 0.000000
0.000000 1.000000 -0.000000
0.000000 -0.000000 1.000000

transpose(p)*a*p - h =
0.000000 -0.000000 -0.000000
0.000000 0.000000 -0.000000
0.000000 0.000000 0.000000
```

Example2
Reduce a complex general matrix to upper Hessenberg form.

```
#include <numeric.h>
int main() {
     int m = 3;

     array double complex a[3][3] = { complex(-149,1), -50, -154,
                              537,  180,  546,
                              -27,   -9,  -25};
     int status;
```

```
    array double complex h[m][m];
    array double complex p[m][m];
    status = hessdecomp(a, h);
    if (status == 0) {
       printf("h =\n%5.2f\n", h);
    }
    else
       printf("Hessenberg matrix calculation error.\n");
    status = hessdecomp(a, h, p);
    if (status == 0) {
       printf("h =\n%5.2f\n", h);
       printf("p =\n%5.2f\n", p);
       printf("transpose(p)*p =\n%5.2f\n", transpose(p)*p);
       printf("conj(transpose(p))*a*p - h =\n%5.2f\n", conj(transpose(p)*a*p - h));
    }
    else
       printf("Hessenberg matrix calculation error.\n");

}
```

Output2

```
h =
complex(-149.00, 1.00) complex(42.20, 0.00) complex(-156.32, 0.00)
complex(-537.68, 0.00) complex(152.55, 0.00) complex(-554.93, 0.00)
complex( 0.00, 0.00) complex( 0.07, 0.00) complex( 2.45, 0.00)

h =
complex(-149.00, 1.00) complex(42.20, 0.00) complex(-156.32, 0.00)
complex(-537.68, 0.00) complex(152.55, 0.00) complex(-554.93, 0.00)
complex( 0.00, 0.00) complex( 0.07, 0.00) complex( 2.45, 0.00)

p =
complex( 1.00, 0.00) complex( 0.00, 0.00) complex( 0.00, 0.00)
complex( 0.00, 0.00) complex(-1.00, 0.00) complex( 0.05, 0.00)
complex( 0.00, 0.00) complex( 0.05, 0.00) complex( 1.00, 0.00)

transpose(p)*p =
complex( 1.00, 0.00) complex( 0.00, 0.00) complex( 0.00, 0.00)
complex( 0.00, 0.00) complex( 1.00, 0.00) complex( 0.00, 0.00)
complex( 0.00, 0.00) complex( 0.00, 0.00) complex( 1.00, 0.00)

conj(transpose(p))*a*p - h =
complex( 0.00,-0.00) complex( 0.00,-0.00) complex( 0.00,-0.00)
complex( 0.00,-0.00) complex( 0.00,-0.00) complex( 0.00,-0.00)
complex( 0.00,-0.00) complex( 0.00,-0.00) complex(-0.00,-0.00)
```

See Also

eigensystem().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

histogram

Synopsis
#include <**numeric.h**>
int histogram(**array double** &*y*, **array double** *x*[&], ... /* [**array double** *hist*[:]] */);

Syntax
histogram(*y, x*)
histogram(*y, x, hist*)

Purpose
Calculate and plot histograms.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
y An input array of data set.

x An input array which contains the bins of the histogram.

hist An output array which contains the calculated data of the histogram.

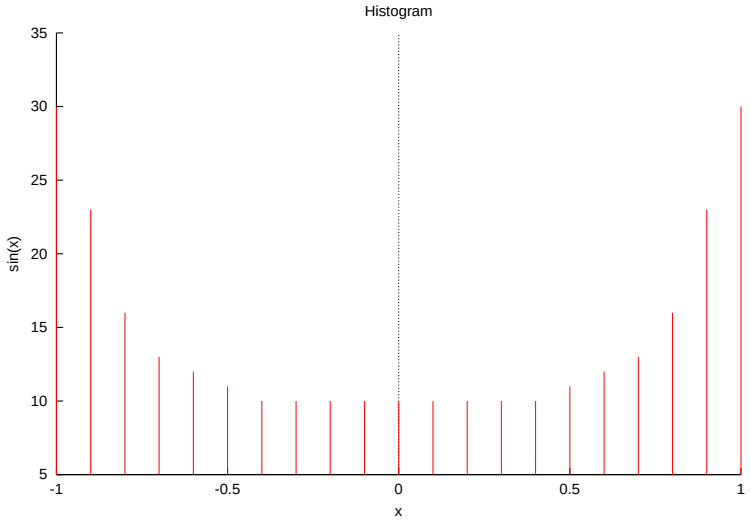
Description
This function calculates and plots the histogram of data set *y* with the bins defined in array *x*. The array of data set *y* can be of any dimensions. If the function is called without the optional argument *hist*, the histogram will be plotted. Otherwise the function only outputs the data of histogram without plotting.

Example1
The histograms of the data set generated from function $\sin(y)$ are calculated and plotted in this example. Note that the histograms are the same for the data set *y1* and *y2* due to the symmetric nature of the $\sin()$ function, even though their periods of the data are different.

```
#define N 300
#define N1 10
#define N2 30
#define M 21

#include<numeric.h>
int main() {
  array double y1[N], x[M], hist[M];
  array double y2[N1][N2];

  linspace(x, -1, 1);
  linspace(y1, 0, 2*M_PI);
  linspace(y2, 0, 4*M_PI);

  y1 = sin(y1);
  y2 = sin(y2);
```

```
  histogram(y1, x); // chplot.histogram
  histogram(y2, x); // chplot.histogram

  histogram(y1, x, hist);
  printf("x hist = %f %f\n", x, hist);
  histogram(y2, x, hist);
  printf("x hist = %f %f\n", x, hist);
}
```

Output1




```
x hist = -1.000000 -0.900000 -0.800000 -0.700000 -0.600000 -0.500000 -0.400000 -0.300000
-0.200000 -0.100000 0.000000 0.100000 0.200000 0.300000 0.400000 0.500000 0.600000
0.700000 0.800000 0.900000 1.000000
 30.000000 23.000000 16.000000 13.000000 12.000000 11.000000 10.000000 10.000000
10.000000 10.000000 10.000000 10.000000 10.000000 10.000000 10.000000 11.000000
12.000000 13.000000 16.000000 23.000000 30.000000

x hist = -1.000000 -0.900000 -0.800000 -0.700000 -0.600000 -0.500000 -0.400000 -0.300000
-0.200000 -0.100000 0.000000 0.100000 0.200000 0.300000 0.400000 0.500000 0.600000
0.700000 0.800000 0.900000 1.000000
 30.000000 23.000000 16.000000 13.000000 12.000000 11.000000 10.000000 10.000000
10.000000 10.000000 10.000000 10.000000 10.000000 10.000000 10.000000 11.000000
12.000000 13.000000 16.000000 23.000000 30.000000
```

Example2

The histograms of the same data sets *y1* and *y2* as in example 1 are calculated in this example. The histograms ara drawn by function **plotxy**() according to the data calculated by function **histogram**(). The histograms are the same as that of example 1.

```
#define N 300
#define N1 10
#define N2 30
#define M 21
```

```
#include <numeric.h>
#include <chplot.h>
int main() {
  array double y1[N], x[M], hist[M];
  array double y2[N1][N2];
  class CPlot plot;

  linspace(x, -1, 1);
  linspace(y1, 0, 2*M_PI);
  linspace(y2, 0, 4*M_PI);

  y1 = sin(y1);
  y2 = sin(y2);

  histogram(y1, x, hist);
  plotxy(x, hist, "Histogram", "x", "sin(x)", &plot);
  plot.axisRange(PLOT_AXIS_Y, min(hist)-5, max(hist)+5);
  plot.plotType(PLOT_PLOTTYPE_IMPULSES, 0);
  plot.plotting();

  histogram(y2, x, hist);
  plotxy(x, hist, "Histogram", "x", "sin(x)", &plot);
  plot.axisRange(PLOT_AXIS_Y, min(hist)-5, max(hist)+5);
  plot.plotType(PLOT_PLOTTYPE_IMPULSES, 0);
  plot.plotting();
}
```

Output2

The output histograms are the same as that of example 1.

See Also

plotxy().

References

householdermatrix

Synopsis
#include <**numeric.h**>
int householdermatrix(**array double complex** *x*[&], **array double complex** *v*[&], ...
/* [**double** **beta*] */);

Syntax
householdermatrix(*x, v*)
householdermatrix(*x, v, beta*)

Purpose
Get the Householder matrix.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Input vector of n elements.

v Output vector of n elements.

beta Optional output value of pointer to double.

Description
This function passes a vector *x* as an input argument and gets the vector *v* and optional output value *beta* such that

$$H = I - \text{beta} * v * v^T$$

where H is a Householder matrix, and I is an identity matrix. A Householder matrix H satisfies the equation

$$H * x = -\text{sign}(x[0]) * \text{norm}(x) * E$$

where vector $E = [1, 0, 0, \cdots, 0]$ has n elements. If *x* is a complex vector, then sign(x[0]) is defined

$$\text{sign}(x[0]) = \frac{x[0]}{\text{abs}(x[0])}$$

Example

```
#include <numeric.h>
int main() {
   array double complex x[5] = {complex(-0.3, 0.5),54,25.3,25.46,83.47};
   array double complex y[5], h[5][5];
   array double x1[5] = {-0.3,54,25.3,25.46,83.47};
   array double e[5]={1,0,0,0,0};
```

```
   array double y1[5], h1[5][5];
   double beta;

   householdermatrix(x1,y1);
   printf("x1=\n%5.2f",x1);
   printf("y1=\n%5.2f",y1);
   householdermatrix(x1,y1,&beta);
   h1 = identitymatrix(5) - beta*y1*transpose(y1);
   printf("y1=\n%5.2f",y1);
   printf("beta=%5.2f\n",beta);
   printf("h1*x1+sign(x1[0])*norm(x1)*e =\n%5.2f",h1*x1+sign(x1[0])*norm(x1,"2")*e);

   householdermatrix(x,y);
   printf("x=\n%5.2f",x);
   printf("y=\n%5.2f",y);
   householdermatrix(x,y,&beta);
   h = identitymatrix(5) - beta*y*conj(transpose(y));
   printf("y=\n%5.2f",y);
   printf("beta=%5.2f\n",beta);
   printf("h*x+sign(x[0])*norm(x)*e =\n%5.2f",h*x+x[0]/abs(x[0])*norm(x,"2")*e);
}
```

Output

```
x1=
-0.30 54.00 25.30 25.46 83.47
y1=
-106.00 54.00 25.30 25.46 83.47
y1=
-106.00 54.00 25.30 25.46 83.47
beta= 0.00
h1*x1+sign(x1[0])*norm(x1)*e =
 0.00  0.00  0.00 -0.00 -0.00
x=
complex(-0.30, 0.50) complex(54.00, 0.00) complex(25.30, 0.00) complex(25.46, 0.00)
complex(83.47, 0.00)
y=
complex(-54.68,91.13) complex(54.00, 0.00) complex(25.30, 0.00) complex(25.46, 0.00)
complex(83.47, 0.00)
y=
complex(-54.68,91.13) complex(54.00, 0.00) complex(25.30, 0.00) complex(25.46, 0.00)
complex(83.47, 0.00)
beta= 0.00
h*x+sign(x[0])*norm(x)*e =
complex( 0.00, 0.00) complex( 0.00,-0.00) complex( 0.00,-0.00) complex( 0.00,-0.00)
complex(-0.00,-0.00)
```

References

G. H. Golub and C. F. van Loan, *Matrix Computations*, third edition, Johns Hopkins University Press, Baltimore, Maryland, 1996

identitymatrix

Synopsis
#include <**numeric.h**>
array double identitymatrix(int *n*)[:][:];

Purpose
Generate an identity matrix.

Return Value
This function returns the identity matrix.

Parameters
n Input integer.

Description
The function returns an $n \times n$ indentity matrix.

Example

```
#include <numeric.h>
int main() {
    int n = 3;
    array double a[n][n];

    a = identitymatrix(n);
    printf("identitymatrix(n) =\n%f\n", a);
}
```

Output

```
identitymatrix(n) =
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
```

See Also
rank().

References

ifft

Synopsis
#include <**numeric.h**>
int ifft(**array double complex** &*y*, **array double complex** &*x*, ... /* [**int** *n* [**int** *dim*[&]]] */);

Syntax
ifft(*y*, *x*)
ifft(*y*, *x*, *n*)
ifft(*y*, *x*, *dim*);

Purpose
N-dimensional inverse Fast Fourier Transform (FFT) calculation.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
y Result of inverse FFT, the array of the same dimension and size as *x*.

x An n-dimensional array used for calculating the inverse FFT.

n Optional argument, user specified inverse FFT points for one-dimensional data.

dim A one-dimensional array of optional arguments with int type. It contains the user specified inverse FFT points. Every element corresponds to a data dimension. For example, there is a three-dimensional data *x[M][N][L]*. The FFT points are specified m, n, l, respectively. Then the array *dim* is given values of $\dim[0] = m, \dim[1] = n, \dim[2] = l$.

Description
The multi-dimensional **array** *x* can be of any supported arithmetic data type and size. Conversion of the data to **double complex** is performed internally. The same multi-dimensional, **double complex array** *y* contains the result of inverse fast Fourier transform. The result normalized by the total number of points of the transform is performed internally. The optional argument *n* or *dim*[] is used for specifying the number of points of the inverse FFT. If the user does not specify the number of points of FFT, it will be initialized according to the length of the input data.

Algorithm
see fft().

Example
see fft().

See Also
fft().

References
R. C. Singleton, *An Algorithm for Computing the Mixed Radix F. F. T.*, IEEE Trans, Audio Electroacoust., AU-1(1969) 93-107.
MJ Oleson, *Netlib Repository at UTK and ORNL*, go/fft-olesen.tar.gz, http://www.netlib.org

integral1

Synopsis
#include <**numeric.h**>
double integral1(**double** (**func*)*(**double**), **double** *x1*, **double** *x2*, ... /* [**double** *tol*] */);

Syntax
integral1(*func*, *x1*, *x2*)
integral1(*func*, *x1*, *x2*, *tol*)

Purpose
Numerical integration of a function.

Return Value
This function returns an integral value of a function *func* integrated from *x1* to *x2*.

Parameters
func Function to be integrated.

x1 An initial point of integral.

x2 A final point of integral.

tol The user specified tolerance. If the user does not specify this optional value, the value of $10*$FLT_EPSILON is used by default, where FLT_EPSILON is defined in header file **float.h**.

Description
The function to be integrated, *func*, is specified as a pointer to a function that takes a **double** as an argument and returns a **double** functional value. *x1* and *x2* are the end-points of the range to be integrated. Conversion of the data to **double** are performed internally. The return value is a **double** data of integral evaluation. If the optional argument *tol* is specified, the integral uses the user specified tolerance to decide iteration stop. Otherwise the default value is used.

Algorithm
The following trapezoidal algorithm for numerical integration of func $f(x)$ is used

$$\int_{x_1}^{x_2} f(x)dx = h[\frac{1}{2}f_1 + \frac{1}{2}f_2] + O(h^3 f'')$$

where f'' represents the second derivative of the function *f*(*x*) evaluated at a point within $x_1 \leq x \leq x_2$.

Example

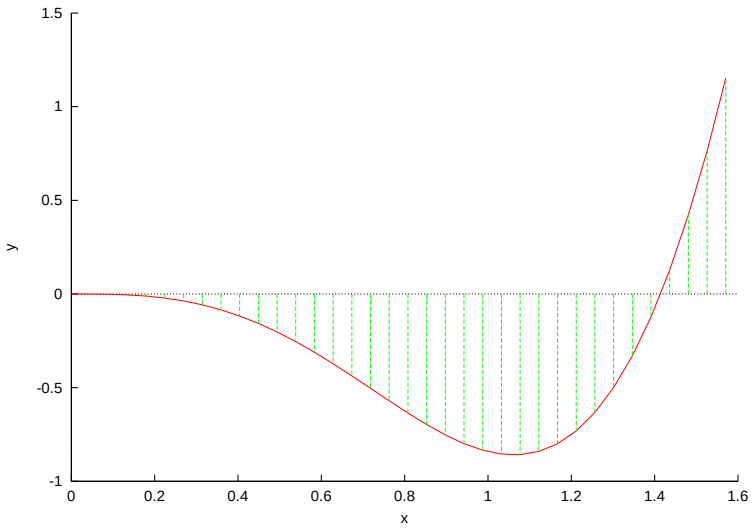
Evaluate the integration

$$\int_0^{\frac{\pi}{2}} x^2(x^2-2)\sin(x)dx$$

```
#include <stdio.h>
#include <math.h>
#include <chplot.h>
#include <numeric.h>

/* Test function */
double func(double x) {
   return x*x*(x*x-2.0)*sin(x);
}

/* Integral of test function */
double fint(double x) {
   return 4.0*x*(x*x-7.0)*sin(x)-(pow(x,4.0)-14.0*x*x+28.0)*cos(x);
}

int main() {
   double x1=0.0,x2=M_PI/2,s;
   int numpoints = 36;
   array double x[numpoints], y[numpoints];
   class CPlot plot;

   linspace(x, x1, x2);
   fevalarray(y, func, x);
   printf("Actual value of the integration is %12.6lf\n",fint(x2)-fint(x1));
   s=integral1(func,x1,x2);
   printf("Result from the function integral1 is %11.6lf\n",s);

   plot.data2D(x, y);
   plot.data2D(x, y);
   plot.plotType(PLOT_PLOTTYPE_IMPULSES, 1);
   plot.plotting();
}
```

Output

```
Actual value of integraion is    -0.479159
Result from the function integration is   -0.479158
```



see also

integral2(), **integral3**(), **integration2**(), **integration3**(), **derivative**().

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press, 1997.

integral2

Synopsis
#include <numeric.h>
double integral2(double (func*)(**double** *x*, **double** *y*), **double** *x1*, **double** *x2* **double** *y1*, **double** *y2*);

Syntax
integral2(*func*, *x1*, *x2*, *y1*, *y2*)

Purpose
Numerical integration of a function of two variables with constant limits for both *x* and *y*.

Return Value
This function returns an integral value of a function *func* integrated from $y1$ to $y2$ in *y* and *x1* to *x2* in *x*.

Parameters
func Function to be integrated.

x1 The lower limit of integration in *x*.

x2 The upper limit of integration in *x*.

y1 The lower limit of integration in *y*.

y2 The upper limit of integration in *y*.

Description
The function to be integrated, *func*, is specified as a pointer to a two-dimensional function that takes two arguments and returns the functional value of double type. The values *x1* and *x2* are the end-points of the range in *x* to be integrated. The values $y1$ and $y2$ are the upper and lower limits in *y*, respectively. The return value is the integral of double type.

Algorithm
The integration is carried out by (1) its lower and upper limits in *y* denoted as *y1* and *y2*; (2) its lower and upper limits in *x* denoted as *x1* and *x2*, respectively, so that

$$I = \int\int dxdyf(x,y) = \int_{x_1}^{x_2} dx \int_{y_1}^{y_2} dyf(x,y)$$

Example
Integrate $f(x,y) = \sin(x)\cos(y) + 1$ over the limits $0 \leq x \leq \pi$ and $-\pi \leq y \leq \pi$ by

$$I = \int_0^{\pi} \int_{-\pi}^{\pi} (\sin(x) * \cos(y) + 1) dxdy$$

```
#include <math.h>
#include <numeric.h>

double func(double x,double y) {
    return sin(x)*cos(y)+1;
}
int main() {
    double x1 = 0, x2 = M_PI;
    double y1 = -M_PI, y2 = M_PI;
    double v;
    v = integral2(func,x1, x2, y1,y2);
    printf("integral2() gives %f\n", v);
}
```

Output

```
integral2() gives 19.739212
```

see also

integral1(), **integral3**(), **integration2**(), **integration3**(), **derivative**().

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press, 1997.

integral3

Synopsis
#include <**numeric.h**>
double integral3(**double** (**func*)(**double** *x*, **double** *y*, **double** it z), **double** *x1*, **double** *x2*, **double** *y1*,
double *y2*, **double** *z1*, **double** *z2*);

Syntax
integration3(*func*, *x1*, *x2*, *y1*, *y2*, *z1*, *z2*)

Purpose
Numerical integration of a function of three-dimensions with constant limits in *x*, *y* and *z*.

Return Value
This function returns an integral value of a function *func* integrated from *z1* to *z2* in *z*, *y1* to *y2* in *y* and *x1* to *x2* in *x*.

Parameters
func Function to be integrated.

x1 The lower limit of integration in *x*.

x2 The upper limit of integration in *x*.

y1 The lower limit of integration in *y*.

y2 The upper limit of integration in *y*.

z1 The lower limit of integration in *z*.

z2 The upper limit of integration in *z*.

Description
The function to be integrated, *func*, is specified as a pointer to a three-dimensional function that takes three arguments and returns a functional value of double type. The values $y1$ and $y2$ are the upper and lower limits in *y*, the values $z1$ and $z2$ are the upper and lower limits in *z*, and the values *x1* and *x2* are the end-points of the range in *x* to be integrated, respectively . The return value is the integral of double type.

Algorithm
The integration is carried out by (1) its lower and upper limits in *z*, denoted *z1* and *z2*, (2)its lower and upper limits in *y*, denoted *y1* and *y2*; and (3)its lower and upper limits in *x*, which we will denote *x1* and *x2*, so that

$$I = \int\int\int dxdydzf(x,y,z) = \int_{x_1}^{x_2} dx \int_{y_1}^{y_2} dy \int_{z_1}^{z_2} dzf(x,y,z)$$

Example

Integrate $f(x, y, z) = \sin(x)\cos(y)\sin(z) + 1$ over the limits of $0 \leq x \leq \pi$, $-\pi \leq y \leq \pi$ and $0 \leq z \leq \pi$.

$$I = \int_0^{\pi} \int_{-\pi}^{\pi} \int_0^{\pi} (\sin(x)\cos(y)\sin(z) + 1) dx dy dz$$

```
#include <math.h>
#include <numeric.h>

double func(double x,double y, double z) {
    return sin(x)*cos(y)*sin(z)+1;
}
int main() {
    double x1 = 0, x2 = M_PI;
    double y1 = -M_PI, y2 = M_PI;
    double z1 = 0, z2 = M_PI;
    double v;
    v = integral3(func, x1,x2, y1, y2, z1, z2);
    printf("integral3() gives %f\n", v);
}
```

Output

```
integral3() gives 62.012571
```

see also

integral1(), **integral2**(), **integral3**(), **integration2**(), **integration3**(), **derivative**().

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press, 1997.

integration2

Synopsis
#include <**numeric.h**>
double integration2(**double** (**func*)(**double** *x*, **double** *y*), **double** *x1*, **double** *x2* **double** (**y1*)(**double** *x*),
double (**y2*)(**double** *x*));

Syntax
integration2(*func*, *x1*, *x2*, *y1*, *y2*)

Purpose
Numerical integration of a function of two variables.

Return Value
This function returns an integral value of a function *func* integrated from $y_1(x)$ to $y_2(x)$ and *x1* to *x2* in *x*.

Parameters
func Function to be integrated.

x1 Function for the lower limit of integration in *x*.

x2 Function for the upper limit of integration in *x*.

y1 Function for the lower limit of integration in *y*.

y2 Function for the upper limit of integration in *y*.

Description
The function to be integrated, *func*, is specified as a pointer to a two-dimensional function that takes two arguments and returns the functional value of double type. The values *x1* and *x2* are the end-points of the range in *x* to be integrated. The values $y_1(x)$ and $y_2(x)$ are user-supplied functions for the upper and lower limits in *y*. The return value is the integral of double type.

Algorithm
The integration is carried out by (1) its lower and upper limits in *y*, denoted as *y1(x)* and *y2(x)*; (2) its lower and upper limits in *x*, denoted as *x1* and *x2*, respectively, so that

$$I = \int\int dxdyf(x,y) = \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} dyf(x,y)$$

Now we can define a function *H(x,y)* that does the innermost integral,

$$H(x,y) = \int_{y_1(x)}^{y_2(x)} f(x,y)dy$$

and finally our answer as an integral over *H(x)*

$$I = \int_{x_1}^{x_2} H(x)dx$$

Example

Integrate r^2 over a circular area with a radius of $r = 2$.

$$I = \int_{-r}^{r} \int_{-\sqrt{r^2-x^2}}^{\sqrt{r^2-x^2}} (x^2 + y^2)dxdy = \int_0^r \int_0^\pi \rho^2 d\theta d\rho = \frac{2\pi r^4}{4}$$

```
#include <stdio.h>
#include <math.h>
#include <numeric.h>

double r = 2;
double func(double x,double y) {
   return x*x+y*y;
}
double y1(double x) {
   return -sqrt(r*r-x*x);
}
double y2(double x) {
   return sqrt(r*r-x*x);
}

int main() {
  double x1=-r, x2=r, s;
  s=integration2(func,x1,x2, y1,y2);
  printf("integration2() = %f\n", s);
  printf("actual integral = %f\n", M_PI*pow(r,4.0)/2.0);
}
```

Output

```
integration2() = 25.156167
actual integral = 25.132744
```

see also

integral1(), **integral2**(), **integral3**(), **integration3**(), **derivative**().

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press, 1997.

integration3

Synopsis
#include <**numeric.h**>
double integration3(**double** (**p*func*)(**double** *x*, **double** *y*, **double** it z), **double** *x1*, **double** *x2*,
double (**y1*)(**double** *x*), **double** (**y2*)(**double** *x*),
double(**z1*)(**double** *x*, **double** *y*), **double**(*z2*)(**double** *x*, **double** *y*));

Syntax
integration3(*func*, *x1*, *x2*, *y1*, *y2*, *z1*, *z2*)

Purpose
Numerical integration of a function of three-dimensions.

Return Value
This function returns an integral value of a function *func* integrated from *z1* to *z2* in *z*, *y1* to *y2* in *y* and *x1* to *x2* in *x*.

Parameters
func Function to be integrated.

x1 Function for the lower limit of integration in *x*.

x2 Function for the upper limit of integration in *x*.

y1 Function for the lower limit of integration in *y*.

y2 Function for the upper limit of integration in *y*.

z1 Function for the lower limit of integration in *z*.

z2 Function for the upper limit of integration in *z*.

Description
The function to be integrated, *func*, is specified as a pointer to a three-dimensional function that takes three arguments and returns a functional value of double type. The values $y_1(x)$ and $y_2(x)$ are user-supplied functions for the upper and lower limits in *y*, the values $z_1(x)$ and $z_2(x)$ are user-supplied functions for the upper and lower limits in *z*, and the values *x1* and *x2* are the end-points of the range in *x* to be integrated, respectively. The return value is the integral of double type.

Algorithm
The integration is carried out by (1) its lower and upper limits in *z* at specified *x* and *y*, denoted *z1(x,y)* and *z2(x,y)*, (2)its lower and upper limits in *y*, denoted *y1(x)* and *y2(x)*; (3)its lower and upper limits in *x*, which

we will denote *x1* and *x2*, so that

$$I = \int\int\int dxdydzf(x,y,z) = \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} dy \int_{z_1(x,y)}^{z_2(x,y)} dzf(x,y,z)$$

Now we can define a function *G(x,y)* that does the innermost integral,

$$G(x,y) = \int_{z_1(x,y)}^{z_2(x,y)} f(x,y,z)dz$$

and a function *H(x)* that does the integral of *G(x,y)*,

$$H(x,y) = \int_{y_1(x)}^{y_2(x)} G(x,y)dy$$

and finally our answer as an integral over *H(x)*

$$I = \int_{x_1}^{x_2} H(x)dx$$

Example

Integrate r^2 over a spherical volume with a radius of $r = 2$.

$$I = \int_{-r}^{r} \int_{-\sqrt{r^2-x^2}}^{\sqrt{r^2-x^2}} \int_{-\sqrt{r^2-x^2-y^2}}^{\sqrt{r^2-x^2-y^2}} (x^2+y^2+z^2)dxdydz = \int_0^{2\pi} \int_0^{\pi} \int_0^{r} \rho^4 \sin(\theta) d\rho d\theta d\phi = \frac{4\pi r^5}{5}$$

```
#include <stdio.h>
#include <math.h>
#include <numeric.h>

double r;
double func(double x,double y,double z) {
   return x*x+y*y+z*z;
}
double y1(double x) {
   return -sqrt(r*r-x*x);
}
double y2(double x) {
   return sqrt(r*r-x*x);
}
double z1(double x,double y) {
   return -sqrt(r*r-x*x-y*y);
}
double z2(double x,double y) {
   return sqrt(r*r-x*x-y*y);
}

int main() {
   double x1=-2, x2= 2, s;
   r = 2;
   s=integration3(func,x1,x2,y1,y2,z1,z2);
   printf("integration3() = %f\n", s);
   printf("actual integral = %f\n", 4.0*M_PI*pow(r,5.0)/5.0);
}
```

Output

```
integration3() = 80.486994
actual integral = 80.424781
```

see also

integral1(), **integral2**(), **integral3**(), **integration2**(), **derivative**().

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press, 1997.

interp1

Synopsis
#include <**numeric.h**>
int interp1(**double** *y*[&], **double** *x*[&], **double** *xa*[&], **double** *ya*[&], **char** **method*);

Purpose
This function finds the values of the function, which is expressed in terms of two arrays *xa* and *ya*, at points expressed in array *x* by linear or cubic spline interpolation.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
y An output array of function values.

x An input array of variable values.

ya An array which contains function values corresponding to *xa*.

xa An array which contains tabulated x values of the function.

method A string which specifies the method of interpolation. The string `"linear"` or `"spline"` is for linear or spline interpolation, respectively.

Description
This function takes two arrays *xa* and *ya* of the same length that express a known function, and calculates the values, at some other points *x* by linear or cubic spline interpolation. The interpolation method can be chosen by input argument *method* of string with `"linear"` or `"spline"`.

Example
Calculate function values of $\sin(x)$ by linear and cubic spline interpolation.

```
#include <stdio.h>
#include <numeric.h>
#define N 10
#define NUM_X 5

int main() {
    int i,nfunc;
    array double f[NUM_X],x[NUM_X],y[NUM_X];
    array float xa[N],ya[N];

    /* Generation of interpolation tables of sin(x) function */
    linspace(xa, 0, M_PI);
    ya = sin(xa);
    linspace(x, 0.15, M_PI-0.15);
    f=sin(x);
```

```
    /*Spline interpolation */
    printf("\nCubic spline interpolation of function sin(x)\n");
    printf("\n%9s %13s %17s\n","x","f(x)","interpolation");
    interp1(y, x, xa, ya, "spline");
    for(i=0;i<NUM_X;i++)
        printf("%12.6f %12.6f %12.6f\n",x[i],f[i],y[i]);

    /*linear interpolation */
    printf("\nlinear interpolation of function sin(x)\n");
    printf("\n%9s %13s %17s\n","x","f(x)","interpolation");
    interp1(y, x, xa, ya, "linear");
    for(i=0;i<NUM_X;i++)
        printf("%12.6f %12.6f %12.6f\n",x[i],f[i],y[i]);
}
```

Output

```
Cubic spline interpolation of function sin(x)

        x          f(x)     interpolation
    0.150000     0.149438     0.149431
    0.860398     0.758102     0.758072
    1.570797     1.000000     0.999960
    2.281195     0.758102     0.758072
    2.991593     0.149438     0.149430

linear interpolation of function sin(x)

        x          f(x)     interpolation
    0.150000     0.149438     0.146972
    0.860398     0.758102     0.746562
    1.570797     1.000000     0.984808
    2.281195     0.758102     0.746562
    2.991593     0.149438     0.146972
```

See Also

interp2(), **CSpline**::**Interp**(), **CSpline**::**Interpm**().

References

William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery, *Numeric Recipes in C*, Second Edition, Cambridge University Press, 1997.

interp2

Synopsis
#include <**numeric.h**>
int interp2(**double** *z*[&][&], **double** *x*[&], **double** *y*[&], **double** *xa*[&], **double** *ya*[&],
double *za*[&][&], **char** **method*);

Purpose
This function finds the values of a two-dimensional function, at points indicated by two-one-dimensional arrays *x* and *y*, by two dimensional linear or cubic spline interpolation. The function is expressed in terms of tabulated values.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
z An output two-dimensional matrix which contains calculated function values

x An array of *x* values at which the function values will be calculated by interpolation.

y An array of *y* values at which the function values will be calculated by interpolation.

xa An input one-dimensional array which contains tabulated values of the first variable.

ya An input one-dimensional array which contains tabulated values of the second variable.

za An input two-dimensional matrix which contains function values corresponding to *xa* and *ya*.

method A string which specifies the method of interpolation. The string `"linear"` and `"spline"` is for linear or spline interpolation, respectively.

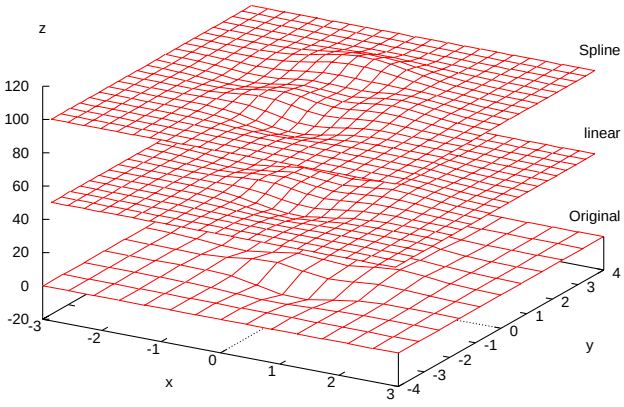
Description
This function takes two arrays, *xa* of dimension *m*, and *ya* of dimension *n*, and a matrix of function value *za* of dimension $m \times n$ tabulated at the grid points defined by *xa* and *ya*, and calculates the values of the function, at points defined by arrays of *x* and *y* by linear or cubic spline interpolation. The dimensions for arrays *xa*, *ya*, *x* and *y* can be different.

Example
Interpolate a two- dimensional function
$f(x,y) = 3(1-x)^2 e^{-x^2-y^2+1} - 10(\frac{x}{5} - x^3 - y^5)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$
by linear and cubic spline interpolations.

```
#include <stdio.h>
#include <chplot.h>
#include <numeric.h>
```

```
#define M 15
#define N 11
#define NUM_X 22
#define NUM_Y 22

int main() {
    int i,j;
    array double z_s1[NUM_X*NUM_Y], z_l1[NUM_X*NUM_Y];
    array double x[NUM_X],y[NUM_Y],xa[M],ya[N];
    array double za[M][N],za1d[M*N], z_s[NUM_X][NUM_Y], z_l[NUM_X][NUM_Y];
    class CPlot plot;

    /* Construct data set of the peaks function */
    linspace(xa, -3, 3);
    linspace(ya, -4, 4);
    for(i=0; i<M; i++) {
        for(j=0; j<N; j++) {
            za[i][j] = 3*(1-xa[i])*(1-xa[i])*
                       exp(-(xa[i]*xa[i])-(ya[j]+1)*(ya[j]+1))
                       - 10*(xa[i]/5 - xa[i]*xa[i]*xa[i]-
                       pow(ya[j],5))*exp(-xa[i]*xa[i]-ya[j]*ya[j])
                       - 1/3*exp(-(xa[i]+1)*(xa[i]+1)-ya[j]*ya[j]);
        }
    }
    za1d = (array double[M*N])za;

    linspace(x, -2.9, 2.9);
    linspace(y, -3.9, 3.9);

    /* test 2-dimensional cubic spline interpolation*/
    interp2(z_s,x,y,xa,ya,za,"spline");

    /* test 2-dimensional linear interpolation */
    interp2(z_l,x,y,xa,ya,za,"linear");

    /* add offset for display */
    z_l1 = (array double[NUM_X*NUM_Y])z_l + (array double[NUM_X*NUM_Y])50;
    z_s1 = (array double[NUM_X*NUM_Y])z_s + (array double[NUM_X*NUM_Y])100;

    plot.data3D(xa, ya, za1d);
    plot.data3D(x, y, z_s1);
    plot.data3D(x, y, z_l1);
    plot.label(PLOT_AXIS_X, "x");
    plot.label(PLOT_AXIS_Y, "y");
    plot.label(PLOT_AXIS_Z, "z");
    plot.ticsLevel(0);
    plot.text("Spline", PLOT_TEXT_RIGHT,3.5,3.5,120);
    plot.text("linear", PLOT_TEXT_RIGHT,3.5,3.5,70);
    plot.text("Original", PLOT_TEXT_RIGHT,3.5,3.5,20);
    plot.plotType(PLOT_PLOTTYPE_LINES,0,1,1);
    plot.plotType(PLOT_PLOTTYPE_LINES,1,1,1);
    plot.plotType(PLOT_PLOTTYPE_LINES,2,1,1);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
}
```

Output



See Also

interp1(), **CSpline**::Interp(), **CSpline**::Interpm().

References

William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery, *Numeric Recipes in C*, Second Edition, Cambridge University Press, 1997.

inverse

Synopsis
#include <**numeric.h**>
array double inverse(**array double** *a*[:][:], ... /* [**int** **status*] */)[:][:];

Syntax
inverse(*a*)
inverse(*a*, *status*)

Purpose
Calculate the inverse of a square matrix of double type.

Return Value
This function returns the inverse matrix.

Parameters
a Input square matrix.

status Output integer indicating the status of calculation.

Description
This function calculates the inverse matrix of a square matrix. If calculation is successful,$status = 0$, otherwise $status \neq 0$. To calculate the inverse of a matrix of complex numbers, use function **cinverse**().

Example1

```
#include <numeric.h>
int main() {
    array double a[2][2]  = {2, 4,
                             3, 7};
    array double inv[2][2];
    int status;

    inv = inverse(a);
    printf("inverse(a) =\n%f\n", inv);
    printf("inverse(a)*a =\n%f\n", inv*a);
    printf("a*inverse(a) =\n%f\n", a*inv);

    inv = inverse(a, &status);
    if(status == 0)
        printf("inverse(a, &status) = %f\n", inv);
    else
        printf("error: numerical error in inverse()\n");
}
```

Output1

```
inverse(a) =
3.500000 -2.000000
-1.500000 1.000000

inverse(a)*a =
1.000000 0.000000
0.000000 1.000000

a*inverse(a) =
1.000000 0.000000
-0.000000 1.000000

inverse(a, &status) = 3.500000 -2.000000
-1.500000 1.000000
```

Example2

```
#include <numeric.h>
array double func(array double B[2][2])[2][2] {
  return inverse(B);
}

int main() {
    array double a[2][2]  = {2, 4,
                             3, 7};
    array double inv[2][2];
    int status;

    inv = func(a);
    printf("inverse(a) =\n%f\n", inv);
    printf("inverse(a)*a =\n%f\n", inv*a);
    printf("a*inverse(a) =\n%f\n", a*inv);
}
```

Output2

```
inverse(a) =
3.500000 -2.000000
-1.500000 1.000000

inverse(a)*a =
1.000000 0.000000
0.000000 1.000000

a*inverse(a) =
1.000000 0.000000
-0.000000 1.000000
```

See Also

cinverse(), **ludecomp**(), **linsolve**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadephia, 1992.

lcm()

Synopsis
#include <**numeric.h**>
int lcm(**array int** &*g*, **array int** &*u*, **array int** &*u*);

Syntax
lcm(*g*, *u*, *v*)

Purpose
Obtain the least common multiple of the corresponding elements of two arrays of integer type.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

g An output array which is the same dimension and size as *u*. It contains the result of the least common multiple calculation of *u* and *v*.

u Any-dimensional array which contains positive integer elements for the least common multiple calculation.

v An array which is the same dimension and size as *u*. It contains another positive integer elements for the lease common multiple calculation.

Description
This function calculates the least common multiple of corresponding elements *u* and *v*. The arrays *u* and *v* shall be the same size and contains positive integer values. The output array *g* is a positive integer array of same size as *u*.

Example

```
#include <numeric.h>
int main() {
   array int u[2][3] = {1,2,7,15,3,4};
   array int v[2][3] = {2,4,8,3,8,3};
   array int g[2][3];
   lcm(g,u,v);
   printf("g=%d",g);
}
```

Output

```
g=2 4 56
15 24 12
```

See Also
gcd().

References
Knuth, Donald Ervin, *The art of computer programming*, Vol. 2, Addison-Wesley Pub. Co., 1973.

lindata

Synopsis
#include <**numeric.h**>
int lindata(**double** *first*, **double** *last*, ... /* [[array] type a[:]...[:]] [type *a, int n] */);
Syntax
lindata(*first*, *last*, *a*);
lindata(*first*, *last*, *a*, *n*);

Purpose
Get linearly spaced data for an array passed from the third argument.

Return Value
This function returns the number of elements in the passed array *a*.

Parameters
first Starting point of the linearly spaced array.

last Ending point of the linearly spaced array.

a An output passed from the third argument with linearly spaced data. It shall be an array or pointer to arithmetic type.

n An input passed from the fourth argument for the number of elements.

Description
This function obtains a linearly spaced values starting with *first* and ending with *last* for the array *a* passed from the third argument. The number of points is taken internally from the passed array *a*. If the third argument is a pointer to arithmetic type, the fourth argument contains the number of elements for the passed object in the third argument.
Example

```
#include <numeric.h>

int main () {
    double a[6], *p;
    array float b[6];
    array int c[6];
    array double aa[6], d[2][3];

    lindata(2, 12, a);
    printf("a[0] = %f\n", a[0]);
    aa = a;
    printf("aa = \n%f\n", aa);
    p = &aa[0];
    lindata(20, 120, p, 6);
    printf("aa = \n%f\n", aa);
    lindata(2, 12, b);
```

```
    printf("b = \n%f\n", b);
    lindata(2, 12, c);
    printf("c = \n%d\n", c);
    lindata(2, 12, d);
    printf("d = \n%f\n", d);
}
```

Output

```
a[0] = 2.000000
aa =
2.000000 4.000000 6.000000 8.000000 10.000000 12.000000

aa =
20.000000 40.000000 60.000000 80.000000 100.000000 120.000000

b =
2.000000 4.000000 6.000000 8.000000 10.000000 12.000000

c =
2 4 6 8 10 12

d =
2.000000 4.000000 6.000000
8.000000 10.000000 12.000000
```

See Also

logdata().

linsolve

Synopsis
#include <**numeric.h**>
int linsolve(**array double** *x*[:], **array double** *a*[:][:], **array double** *b*[:]);

Purpose
Solve the linear system of equations by LU decomposition.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Output array which contains the solution of equations.

a Input two-dimensional array contains the coefficients of a system of linear equations.

b Input one-dimensional array for linear equations.

Description
This function solves the system of linear equations by LU decomposition. The input matrix *a* should be n-by-n. The function returns 0 if there is a solution for the equation set, otherwise it returns -1.

It calculates the solution to a real/complex system of linear equations $a * X = b$, where a is an n-by-n matrix. The LU decomposition with partial pivoting and row interchanges is used to factor *a* as $a = P*L*U$, where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of *a* is then used to solve the system of equation $a * X = b$.

Example1
Solution of a linear system of equations with real coefficients.

```
#include <numeric.h>
int main() {
     array double a[3][3] = {3, 0, 6,
                             0, 2, 1,
                             1, 0, 1};
     array double b[3] = {2,
                          13,
                          25};
     array double x[3];
     int status;

     linsolve(x, a, b);
     printf("x =\n%f\n", x);

     status = linsolve(x, a, b);
     if(status == 0)
         printf("x =\n%f\n", x);
     else
```

```
        printf("error: numerical error in linsolve()\n");
}
```

Output1

```
x =
49.333333 18.666667 -24.333333

x =
49.333333 18.666667 -24.333333
```

See Also

llsqsolve(), **clinsolve**(), **inverse**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

linspace

Synopsis
#include <**numeric.h**>
int linspace(**array double** &*x*, **double** *first*, **double** *last*);

Purpose
Get linearly spaced values for an array.

Return Value
This function returns the number of elements in array *x*.

Parameters
x An output with linearly spaced values.

first Starting point of the linearly spaced array.

last Ending point of the linearly spaced array.

Description
This function obtains a linearly spaced values starting with *first* and ending with *last* for the array *x*. The number of points is taken internally from the array *x*.

Note
linspace() is obsolete. Use **lindata**().

Example

```
#include <numeric.h>
int main() {
  array double x[11], x2[2][3], x3[2][3][2];
  array float y[11];
  array int z[11];
  array unsigned int ui[11];
  array unsigned short uh[11];
  array short h[11];
  array unsigned char uc[11];
  array char c[11];
  int n;

  linspace(x, 1, sizeof(x)/sizeofelement(elementtype(x)));
  printf("x = %f\n", x);

  linspace(x2, 1, sizeof(x2)/sizeofelement(elementtype(x2)));
  printf("x2 = %f\n", x2);

  linspace(x3, 1, sizeof(x3)/sizeofelement(elementtype(x3)));
  printf("x3 = %f\n", x3);
```

```
  linspace(x, 0, M_PI);
  printf("x = %f\n", x);
  n = linspace(x, 0, M_PI);
  printf("n = %d\n", n);

  linspace(y, 0, M_PI); /* y is float */
  printf("y = %f\n", y);

  linspace(z, 0, 90);
  printf("z = %d\n", z);

  linspace(ui, 0, 90);
  printf("ui = %d\n", ui);

  linspace(uh, 0, 90);
  printf("uh = %d\n", uh);

  linspace(h, 0, 90);
  printf("h = %d\n", h);

  linspace(c, 0, 90);
  printf("c = %d\n", c);

  linspace(uc, 0, 90);
  printf("uc = %d\n", uc);
}
```

Output

```
x = 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
10.000000 11.000000

x2 = 1.000000 2.000000 3.000000
4.000000 5.000000 6.000000

x3 = 1.000000 2.000000
3.000000 4.000000
5.000000 6.000000

7.000000 8.000000
9.000000 10.000000
11.000000 12.000000

x = 0.000000 0.314159 0.628319 0.942478 1.256637 1.570797 1.884956 2.199115 2.513274
2.827434 3.141593

n = 11
y = 0.000000 0.314159 0.628319 0.942478 1.256637 1.570796 1.884956 2.199115 2.513274
2.827434 3.141593

z = 0 9 18 27 36 45 54 63 72 81 90

ui = 0 9 18 27 36 45 54 63 72 81 90

uh = 0 9 18 27 36 45 54 63 72 81 90

h = 0 9 18 27 36 45 54 63 72 81 90

c = 0 9 18 27 36 45 54 63 72 81 90
```

```
uc = 0 9 18 27 36 45 54 63 72 81 90
```

See Also
logspace().

References

llsqcovsolve()

Synopsis
#include <**numeric.h**>
int llsqcovsolve(**array double** *x[&]*, **array double** *a[&][&]*, **array double** *b[&]*,
array double *v[&][&]*, ... /* [**array double** *p[&]*] */);

Syntax
llsqcovsolve(*a*, *b*, *v*, *x*)
llsqcovsolve(*a*, *b*, *v*, *x*, *p*)

Purpose
Solve linear system of equations based on linear least-squares with known covariance.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
a A two-dimensional array size of $m \times n$ which contains input data. It is an overdetermined least-squares problem. So m must be larger than n.

b A vector with m elements. It contains input data.

v An input array of size $m \times m$. It contains covariance value.

x An output vector with n elements. It contains the linear least-squares results.

p An optional output vector with n elements. It contains the standard errors of *x*.

Description
This function calculates the vector *x* that satisfies $a * x = b + e$ where e is a normally distributed error with zero mean and covariance *v* using the linear least-squares method. This is an overdetermined linear least-squares problem. So the number of rows m must be larger than the number of column n. If optional output vector *p* is specified, it will pass out the standard errors of *x*.

Algorithm
The vector *x* can be calculated through the minimizes the quantity $(ax - b)^T v^{-1}(ax - b)$. The classical linear algebra solution to this problem is

$$x = (a^T v^{-1} a)^{-1} a^T v^{-1} b$$

and the standard errors of *x* is

$$\begin{aligned} \text{mse} &= b^T(v^{-1} - v^{-1}a(a^T v^{-a}a)^{-1}a^T v^{-1})b/(m-n) \\ p &= \sqrt{\textbf{diagonal}(a^T v^{-1} a)^{-1} * \text{mse}} \end{aligned}$$

Example

```
#include <numeric.h>

int main() {
     array double a[4][3] = {1, 2, 3,
                             4, 5, 6,
                             7, 8, 9,
                             11,12,3}; /* a[m][n] */
     array double b[4] = {4,4,54,6};
     array double v[4][4] ={ 1, 1, 1, 1,
                             1, 2, 3, 4,
                             1, 3, 6,10,
                             1, 4,10,20};
     array double x[3], dx[3];

     llsqcovsolve(x, a, b, v, dx);
     printf("a = \n%f", a);
     printf("b = \n%f", b);
     printf("v = \n%f", v);
     printf("x = \n%f", x);
     printf("a*x = \n%f", a*x);
     printf("dx = \n%f", dx);
}
```

Output

```
a =
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000
11.000000 12.000000 3.000000
b =
4.000000 4.000000 54.000000 6.000000
v =
1.000000 1.000000 1.000000 1.000000
1.000000 2.000000 3.000000 4.000000
1.000000 3.000000 6.000000 10.000000
1.000000 4.000000 10.000000 20.000000
x =
10.800000 -25.600000 14.800000
a*x =
4.000000 4.000000 4.000000 -144.000000
dx =
61.237244 54.772256 5.270463
```

See Also

qrdecomp(), **llsqnonnegsolve**(), **llsqsolve**().

References

Franklin A. Graybill, *Theory and Application of the Linear Model*, Duxbury Press, 1976.
Gilbert Strang, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, 1986.

llsqnonnegsolve()

Synopsis
#include <**numeric.h**>
int llsqnonnegsolve(array double *x[&]*, **array double** *a[&][&]*, **array double** *b[&]*, ...
/* [**double** tol, **array double** *w[&]*] */);

Syntax
llsqnonnegsolve(*a*, *b*, *x*)
llsqnonnegsolve(*a*, *b*, *x*, *tol*)
llsqnonnegsolve(*a*, *b*, *x*, *tol*, *p*)

Purpose
Solve linear system of equation with non-negative values based on linear least-squares method.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
a A two-dimensional array size of $m \times n$ which contains input data.

b A vector with m elements. It contains input data.

x An output vector with n elements. It contains the non-negative least squares results.

tol An optional input which specifies the tolerance of the solution. If the user does not specify this argument or specify zero, $tol = 10 * max(m, n) * norm(u, "1") *$ FLT_EPSILON is used by default. FLT_EPSILON is defined in header file **float.h**.

w An optional output vector with n elements. It contains a vector where $w[i] < 0$ when $x[i] = 0$ and $w[i] \cong 0$ when $x[i] > 0$.

Description
This function solves the equations $ax = b$ using the linear least-squares method, while it is subject to the constraint that the solution vector *x* has non-negative elements. That is, $x[i] \geq 0$ for $i = 0, 1, \cdots, n - 1$.

Algorithm
This function uses the algorithm described in the reference book, Chapter 23.

Example

```
#include <numeric.h>
int main () {
```

```
array double a[15][5]={-0.1340555,-0.2016283,-0.1693078,-0.1897199,-0.1738723,
                       -0.1037948,-0.1576634,-0.1334626,-0.1484855,-0.1359769,
                       -0.0877960,-0.1288387,-0.1068301,-0.1201180,-0.1093297,
                       0.02058554,0.00335331,-0.0164127,0.00078606,0.00271659,
                       -0.0324809,-0.0187680,0.00410639,-0.0140589,-0.0138439,
                       0.05967662,0.06667714,0.04352153,0.05740438,0.05024962,
                       0.06712457,0.07352437,0.04489770,0.06471862,0.05876445,
                       0.08687186,0.09368296,0.05672327,0.08141043,0.07302320,
                       0.02149662,0.06222662,0.07213486,0.06200069,0.05570931,
                       0.06687407,0.10344506,0.09153849,0.09508223,0.08393667,
                       0.15879069,0.18088339,0.11540692,0.16160727,0.14796479,
                       0.17842887,0.20361830,0.13057860,0.18385729,0.17005549,
                       0.11414080,0.17259611,0.14816471,0.16007466,0.17005549,
                       0.07846038,0.14669563,0.14365800,0.14003842,0.12571277,
                       0.10803175,0.16994623,0.14971519,0.15885312,0.14301547},
    b[15]={-0.4361,-0.3437,-0.2657,-0.0392,0.0193,0.0747,0.0935,0.1079,0.1930,
           0.2058,0.2606,0.3142,0.3529,0.3615,0.3647};
   array double x[5], w[5];
   llsqnonnegsolve(x,a,b,0.0,w);
   printf("a=%6.4f",a);
   printf("b=%6.4f",b);
   printf("x=%6.4f",x);
   printf("w=%6.4f",w);
}
```

Output

```
a=-0.1341 -0.2016 -0.1693 -0.1897 -0.1739
-0.1038 -0.1577 -0.1335 -0.1485 -0.1360
-0.0878 -0.1288 -0.1068 -0.1201 -0.1093
0.0206 0.0034 -0.0164 0.0008 0.0027
-0.0325 -0.0188 0.0041 -0.0141 -0.0138
0.0597 0.0667 0.0435 0.0574 0.0502
0.0671 0.0735 0.0449 0.0647 0.0588
0.0869 0.0937 0.0567 0.0814 0.0730
0.0215 0.0622 0.0721 0.0620 0.0557
0.0669 0.1034 0.0915 0.0951 0.0839
0.1588 0.1809 0.1154 0.1616 0.1480
0.1784 0.2036 0.1306 0.1839 0.1701
0.1141 0.1726 0.1482 0.1601 0.1701
0.0785 0.1467 0.1437 0.1400 0.1257
0.1080 0.1699 0.1497 0.1589 0.1430
b=-0.4361 -0.3437 -0.2657 -0.0392 0.0193 0.0747 0.0935 0.1079 0.1930 0.2058 0.2606
0.3142 0.3529 0.3615 0.3647
x=0.0000 0.0000 2.4393 0.0000 0.0000
w=-0.0055 -0.0035 0.0000 -0.0024 -0.0023
```

See Also

llsqcovsolve(), **llsqsolve**().

References

Lawson, C. L. and R. J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, 1974.

llsqsolve

Synopsis
#include <**numeric.h**>
int llsqsolve(**array double complex** *x*[&], **array double complex** *a*[&][&],
array double complex *b*[&]);

Purpose
Linear least-squares solution of a linear system of equations.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Output array which contains the solution of a linear system of equations.

a Input matrix for a linear system of equations.

b Input array for constants of a linear system of equations.

Description
This function solves the linear system of equations by the method called *linear least-squares* method. The input matrix *x* can be of different number of rows and columns. But, the number of elements for *x* shall be the same as the number of columns in matrix *a*. The number of elements for *b* shall be the same as the number of rows in matrix *a*. The function can handle the equation set with complex numbers. The function returns 0 if there is a solution for the system of linear equations, otherwise it returns -1.
Algorithm
The linear system of equations

$$a * X = b$$

is solved by minimizing the squared error in $(a * X - b)$.

Example1
Solutions of linear system of equations with *a*[3][3] and *a*[3][4].

```
#include <numeric.h>
#define N1 3
#define N2 4
int main() {
     array double a[N1][3] = {3, 0, 6,
                              0, 2, 1,
                              1, 0, 1};
     array double a2[N2][3] = {3, 0, 6,
                               0, 2, 1,
                               1, 0, 1,
```

```
                              4, 5, 2};
     array double b[N1] = {2,
                           13,
                           25};
     array double b2[N2] = {2,
                            13,
                            25,
                            1};
     array double x[3], x2[3];
     int status;

     llsqsolve(x, a, b);
     printf("llsqsolve(x,a,b) = \n%f\n", x);
     printf("a*x = \n%f\n", a*x);
     llsqsolve(x2, a2, b2);
     printf("llsqsolve(x2, a2,b2) = \n%f\n", x2);
     printf("a2*x2 = \n%f\n", a2*x2);

     status = llsqsolve(x, a, b);
     if(status == 0)
         printf("llsqsolve(x, a,b) =\n%f\n", x);
     else
         printf("error: numerical error in llsqsolve()\n");
}
```

Output1

```
llsqsolve(x,a,b) =
49.333333 18.666667 -24.333333

a*x =
2.000000 13.000000 25.000000

llsqsolve(x2, a2,b2) =
-2.256881 1.715596 2.198777

a2*x2 =
6.422018 5.629969 -0.058104 3.948012

llsqsolve(x, a,b) =
49.333333 18.666667 -24.333333
```

Example2
Solution of linear system of equations with complex numbers.

```
#include <numeric.h>
int main() {
     array double complex a[3][3] = {complex(3,2), 0, 6,
                              0, 2, 1,
                              1, 0, 1};
     array double complex a2[3][4] = {complex(3,2), 0, 6, 0,
                                      2, 1, 1, 0,
                                      1, 4, 5, 2};
     array double complex b[3] = {2,
                           13,
                           25};
     array double complex b2[3] = {2,
                           13,
                           25};
```

```
    array double complex x[3], x2[4];
    int status;

    llsqsolve(x, a, b);
    printf("llsqsolve(x, a,b) = \n%f\n", x);
    llsqsolve(x2, a2, b2);
    printf("a*x = \n%f\n", a*x);
    printf("llsqsolve(x2, a2,b2) = \n%f\n", x2);
    printf("a2*x2 = \n%f\n", a2*x2);

    status = llsqsolve(x, a, b);
    if(status == 0)
        printf("llsqsolve(x,a,b) is ok\n");
    else
        printf("error: numerical error in llsqsolve()\n");
}
```

Output2

```
llsqsolve(x, a,b) =
complex(34.153846,22.769231) complex(11.076923,11.384615) complex(-9.153846,-22.769231)


a*x =
complex(2.000000,0.000000) complex(13.000000,0.000000) complex(25.000000,0.000000)


llsqsolve(x2, a2,b2) =
complex(3.953216,0.102924) complex(6.702534,1.163353) complex(-1.608967,-1.369201)
complex(1.140741,1.044834)

a2*x2 =
complex(2.000000,0.000000) complex(13.000000,-0.000000) complex(25.000000,0.000000)


llsqsolve(x,a,b) is ok
```

See Also

linsolve().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

logm

Synopsis
#include <**numeric.h**>
int logm(**array double complex** *y[&][&]*, **array double complex** *x[&][&]*);

Syntax
logm(*y*, *x*)

Purpose
Computes the matrix natural logarithm.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Input square matrix. It contains data to be calculated.

y Output square matrix which contains data of the result of *x* natural logarithm..

Description
This function computes the matrix natural logarithm of the input matrix *x*. It is the inverse function of **expm**(). The complex results of *y* will be produced if *x* has negative eigenvalues.

Example
This example shows the usage of **expm**() and **logm**(). A real or complex matrix will change back to itself when a matrix is calculated by **expm**() and then followed by **logm**().

```
#include <numeric.h>
int main() {
   array double x[3][3]={1,2,3,
                         3,4,5,
                         6,7,8};
   array double complex zx[3][3]={complex(1,1),complex(2,2),0,
                         3,complex(4,1),complex(2,5),
                         0,0,0};
   array double complex zy[3][3];
   array double complex y[3][3];

   expm(y,x);
   printf("x = \n%f",x);
   printf("y = \n%5.3f",y);
   logm(x,y);
   printf("x = \n%f",x);
   printf("\n");

   expm(zy,zx);
   printf("zx = \n%5.3f",zx);
```

```
   printf("zy = \n%5.3f",zy);
   logm(zx,zy);
   printf("zx = \n%5.3f",zx);
}
```

Output

```
x =
1.000000 2.000000 3.000000
3.000000 4.000000 5.000000
6.000000 7.000000 8.000000
y =
complex(157372.953,0.000) complex(200034.605,0.000) complex(242697.257,0.000)

complex(290995.910,0.000) complex(369883.552,0.000) complex(448769.194,0.000)

complex(491431.846,0.000) complex(624654.473,0.000) complex(757878.099,0.000)

x =
1.000000 2.000000 3.000000
3.000000 4.000000 5.000000
6.000000 7.000000 8.000000

zx =
complex(1.000,1.000) complex(2.000,2.000) complex(0.000,0.000)
complex(3.000,0.000) complex(4.000,1.000) complex(2.000,5.000)
complex(0.000,0.000) complex(0.000,0.000) complex(0.000,0.000)
zy =
complex(-44.901,56.755) complex(-87.478,70.853) complex(-101.511,-18.910)

complex(-12.469,118.748) complex(-57.370,175.502) complex(-155.470,67.839)

complex(0.000,0.000) complex(0.000,0.000) complex(1.000,0.000)
zx =
complex(1.000,1.000) complex(2.000,2.000) complex(0.000,0.000)
complex(3.000,0.000) complex(4.000,1.000) complex(2.000,5.000)
complex(0.000,0.000) complex(0.000,0.000) complex(0.000,0.000)
```

See Also

expm(), **funm**(), **cfunm**(), **sqrtm**().

References

G. H. Golub, C. F. Van Loan, Matrix Computations Third edition, The Johns Hopkins University Press, 1996

logdata

Synopsis
#include <**numeric.h**>
int logdata(**double** *first*, **double** *last*, ... /* [array] type a[:]...[:] */);

Syntax
logdata(*first*, *last*, *a*);

Purpose
Get logarithmically spaced data for an array passed from the third argument.

Return Value
This function returns the number of elements in the passed array *a*.

Parameters
first Starting point of the logarithmically spaced array.

last Ending point of the logarithmically spaced array.

a An output passed from the third argument with logarithmically spaced data.

Description
This function obtains a logarithmically spaced values starting with *first* and ending with *last* for the array *a* passed from the third argument. The number of points is taken internally from the passed array *a*.

Example

```
#include <numeric.h>

int main () {
    double a[6];
    array float b[6];
    array int c[6];
    array double aa[6], d[2][3];

    logdata(0, 2, a);
    printf("a[0] = %f\n", a[0]);
    aa = a;
    printf("aa = \n%f\n", aa);
    logdata(0, 2, b);
    printf("b = \n%f\n", b);
    logdata(0, 2, c);
    printf("c = \n%d\n", c);
    logdata(0, 2, d);
    printf("d = \n%f\n", d);
}
```

Output

See Also
lindata().

logspace

Synopsis
#include <**numeric.h**>
int logspace(**array double** &*x*, **double** *first*, **double** *last*);

Purpose
Get logarithmically spaced values for an array.

Return Value
This function returns the number of elements in array *x*.

Parameters
x An output array with logarithmically spaced values.

first Starting point of the logarithmically spaced array.

last Ending point of the logarithmically spaced array.

Description
This function obtains a logarithmically spaced values starting with *first* and ending with *last* for array *x*. The number of points is taken internally from the array *x*.

Note
logspace() is obsolete. Use **logdata**().

Example

```
#include <numeric.h>
int main() {
  array double x[11], x2[2][3];
  array float y[11];
  array complex z[11];
  int n;

  logspace(x, 0, 2);
  printf("x = %f\n", x);
  logspace(x2, 0, 2);
  printf("x2 = %f\n", x2);
  n = logspace(x, 0, 2);
  printf("n = %d\n", n);
  n = logspace(y, 0, 2);
  printf("y = %f\n", y);
  logspace(z, 0, 2);
  printf("z = %f\n", z);
}
```

Output

```
x = 1.000000 1.584893 2.511886 3.981072 6.309573 10.000000 15.848932 25.118864
39.810717 63.095734 100.000000

x2 = 1.000000 2.511886 6.309573
15.848932 39.810717 100.000000

n = 11
y = 1.000000 1.584893 2.511886 3.981072 6.309574 10.000000 15.848932 25.118864
39.810719 63.095734 100.000000

z = complex(1.000000,0.000000) complex(1.584893,0.000000) complex(2.511886,0.000000)
complex(3.981072,0.000000) complex(6.309574,0.000000) complex(10.000000,0.000000)
complex(15.848932,0.000000) complex(25.118864,0.000000) complex(39.810719,0.000000)
complex(63.095734,0.000000) complex(100.000000,0.000000)
```

See Also

linspace().

References

ludecomp

Synopsis
#include <**numeric.h**>
int ludecomp(**array double complex** *a*[&][&], **array double complex** *l*[&][&],
array double complex *u*[&][&],... /***array int** *p*[:][:] */);

Syntax
ludecomp(a, l, u)
ludecomp(a, l, u, p)

Purpose
LU decomposition of a general m-by-n matrix.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
a Input matrix.

l Output L matrix.

u Output U matrix.

p Output P matrix.

Description
An LU decomposition of a square matrix *a* is calculated using partial pivoting with row interchanges. The factorization has the form $a = P * L * U$, where P is a permutation matrix, L a lower triangular matrix with unit diagonal elements and U an upper triangular matrix.

Example1
LU decomposition of a real 3×3 matrix.

```
#include <numeric.h>
int main() {
     int n = 3;
     array double a[3][3] = {2,1,-2,
                             4,-1,2,
                             2,-1,1} ; /* n-by-n matrix */
     array double a2[3][3] = {-1,5,6,
                              3,-6,1,
                              6,8, 9} ; /* n-by-n matrix */
     array double l[n][n], u[n][n];
     array int p[n][n];
     int status;

     status = ludecomp(a, l, u);
```

```
    printf("l =\n%f\n", l);
    printf("u =\n%f\n", u);
    printf("l*u =\n%f\n", l*u);

    /* pass the address of p[0][0]
       or use ludecomp(a, l, u, &p);
       or use ludecomp(a, l, u, &p[0][0]); */
    status = ludecomp(a, l, u, p);
    printf("l =\n%f\n", l);
    printf("u =\n%f\n", u);
    printf("p =\n%d\n", p);
    printf("p*l*u =\n%f\n", p*l*u);

    status = ludecomp(a2, l, u);
    if(status == 0) {
        printf("l =\n%f\n", l);
        printf("u =\n%f\n", u);
        printf("l*u =\n%f\n", l*u);
    }
    else
        printf("error: numerical error in ludecomp()\n");

    status = ludecomp(a2, l, u, p);
    printf("l =\n%f\n", l);
    printf("u =\n%f\n", u);
    printf("p =\n%d\n", p);
    printf("p*l*u =\n%f\n", p*l*u);
}
```

Output1

```
l =
0.500000 1.000000 0.000000
1.000000 0.000000 0.000000
0.500000 -0.333333 1.000000

u =
4.000000 -1.000000 2.000000
0.000000 1.500000 -3.000000
0.000000 0.000000 -1.000000

l*u =
2.000000 1.000000 -2.000000
4.000000 -1.000000 2.000000
2.000000 -1.000000 1.000000

l =
1.000000 0.000000 0.000000
0.500000 1.000000 0.000000
0.500000 -0.333333 1.000000

u =
4.000000 -1.000000 2.000000
0.000000 1.500000 -3.000000
0.000000 0.000000 -1.000000

p =
0 1 0
```

```
1 0 0
0 0 1

p*l*u =
2.000000 1.000000 -2.000000
4.000000 -1.000000 2.000000
2.000000 -1.000000 1.000000

l =
-0.166667 -0.633333 1.000000
0.500000 1.000000 0.000000
1.000000 0.000000 0.000000

u =
6.000000 8.000000 9.000000
0.000000 -10.000000 -3.500000
0.000000 0.000000 5.283333

l*u =
-1.000000 5.000000 6.000000
3.000000 -6.000000 1.000000
6.000000 8.000000 9.000000

l =
1.000000 0.000000 0.000000
0.500000 1.000000 0.000000
-0.166667 -0.633333 1.000000

u =
6.000000 8.000000 9.000000
0.000000 -10.000000 -3.500000
0.000000 0.000000 5.283333

p =
0 0 1
0 1 0
1 0 0

p*l*u =
-1.000000 5.000000 6.000000
3.000000 -6.000000 1.000000
6.000000 8.000000 9.000000
```

Example2
LU decomposition of 3×3 matrix with complex number.

```
#include <numeric.h>
int main() {
     int n = 3;
     array double complex a[3][3] = {complex(-1,1),5,6,
                                      3,-6,1,
                                      6,8, 9} ; /* n-by-n matrix */
     array double complex l[n][n], u[n][n];

     ludecomp(a, l, u);
     printf("l =\n%f\n", l);
     printf("u =\n%f\n", u);
     printf("l*u =\n%f\n", l*u);
}
```

Output2

```
l =
complex(-0.166667,0.166667) complex(-0.633333,0.133333) complex(1.000000,0.000000)

complex(0.500000,0.000000) complex(1.000000,0.000000) complex(0.000000,0.000000)

complex(1.000000,0.000000) complex(0.000000,0.000000) complex(0.000000,0.000000)


u =
complex(6.000000,0.000000) complex(8.000000,0.000000) complex(9.000000,0.000000)

complex(0.000000,0.000000) complex(-10.000000,0.000000) complex(-3.500000,0.000000)

complex(0.000000,0.000000) complex(0.000000,0.000000) complex(5.283333,-1.033333)


l*u =
complex(-1.000000,1.000000) complex(5.000000,0.000000) complex(6.000000,0.000000)

complex(3.000000,0.000000) complex(-6.000000,0.000000) complex(1.000000,0.000000)

complex(6.000000,0.000000) complex(8.000000,0.000000) complex(9.000000,0.000000)
```

See Also

linsolve, **inverse**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

maxloc

Synopsis

#include <**numeric.h**>
int maxloc(**array double** &*a*);

Purpose

Find the index of element with the maximum value in an array.

Return Value

This function returns the index for the elements with the maximum value in an array.

Parameters

a The input array in which the maximum value shall be found.

Description

The function finds the maximum element in the array *a*.

Example

```
#include <numeric.h>
int main() {
    double a[3] = {1,2,3};
    double b[2][3] = {1,2,3,10,5,6};
    float  b1[2][3] = {1,2,3,10,5,6};
    int    b2[2][3] = {1,2,3,10,5,6};
    double c[2][3][5];
    c[0][0][3] = 10;
    c[1][0][3] = -10;
    int loc;

    loc = maxloc(a);
    printf("maxloc(a) = %d\n", loc);
    loc = maxloc(b);
    printf("maxloc(b) = %d\n", loc);
    loc = maxloc(b1);
    printf("maxloc(b1) = %d\n", loc);
    loc = maxloc(b2);
    printf("maxloc(b2) = %d\n", loc);
    loc = maxloc(c);
    printf("maxloc(c) = %d\n", loc);
}
```

Output

```
maxloc(a) = 2
maxloc(b) = 3
maxloc(b1) = 3
maxloc(b2) = 3
maxloc(c) = 3
```

See Also
minloc(), **maxv**().

References

maxv

Synopsis

#include <**numeric.h**>
array double maxv(array double *a*[&][&])[:];

Purpose

Find the maximum values of the elements of each row of a matrix.

Return Value

This function returns an array which contains the maximum values of each row of the matrix.

Parameters

a Input matrix.

Description

This function finds the maximum values of each row of a matrix. The matrix can be of any arithmetic data type.

Example

```
#include <numeric.h>
int main() {
    double a[2][3] = {1,2,3,
                      4,5,6};
    array double b[3][4] = {1,2,3,4,
                            5,6,7,8,
                            1,2,3,4};
    array double maxv1[2], maxv2[3];

    maxv1 = maxv(a);
    printf("maxv(a) = %f\n", maxv1);
    maxv2 = maxv(b);
    printf("maxv(b) = %f\n", maxv2);
}
```

Output

```
maxv(a) = 3.000000 6.000000

maxv(b) = 4.000000 8.000000 4.000000
```

See Also

maxloc(), **minloc**().

References

mean

Synopsis
#include <**numeric.h**>
double mean(**array double** &*a*, ... /*[**array double** *v*[:]] */);

Syntax
mean(*a*)
mean(*a*, *v*)

Purpose
Calculate the mean value of all elements of an array and mean values of the elements of each row of a two-dimensional array.

Return Value
This function returns the mean value of all the elements.

Parameters
a An input array of any dimension.

v An output array which contains the mean values of each row of a two-dimensional array.

Description
The function calculates the mean value of all elements in an array of any dimension. If the array is a two-dimensional matrix, the function can calculate mean values of each row. The mean values of each row are passed out by argument *v*.

Example1
Calculate the mean values of all elements of arrays with different dimensions.

```
#include <numeric.h>
int main() {
    double a[3] = {1,2,3};
    int b[2][3] = {1,2,3,4,5,6};
    double c[2][3][5];
    c[0][0][0] = 10;
    c[0][0][1] = 20;
    double meanval;

    meanval = mean(a);
    printf("mean(a) = %f\n", meanval);
    meanval = mean(b);
    printf("mean(b) = %f\n", meanval);
    meanval = mean(c);
    printf("mean(c) = %f\n", meanval);
}
```

Output1

```
mean(a) = 2.000000
mean(b) = 3.500000
mean(c) = 1.000000
```

Example2

Calculate the mean values of all elements of arrays with different dimensions. The mean values of each row are passed by argument *v*.

```
#include <numeric.h>
#define N 2
#define M 3
int main() {
    double a[N][M] = {1,2,3,
                      4,5,6};
    array double b[3][4] = {1,2,3,4,
                            5,6,7,8,
                            1,2,3,4};
    array double meana1[N], meanb1[3];
    double meanval;

    /* Note: second argument of mean() must be double data type */

    meanval = mean(a, meana1);
    printf("meanval = mean(a, meana1) = %f\n", meanval);
    printf("mean(a, meana1) = %f\n", meana1);


    meanval = mean(b, meanb1);
    printf("meanval = mean(b, meanb1) = %f\n", meanval);
    printf("mean(b, meanb1) = %f\n", meanb1);
}
```

Output2

```
meanval = mean(a, meana1) = 3.500000
mean(a, meana1) = 2.000000 5.000000

meanval = mean(b, meanb1) = 3.833333
mean(b, meanb1) = 2.500000 6.500000 2.500000
```

Example3

Calculate the mean values of each column of an array. The calculation is carried out by transposing the array.

```
#include <numeric.h>
int main() {
    double a[2][3] = {1,2,3,
                      4,5,6};
    array double b[3][4] = {1,2,3,4,
                            5,6,7,8,
                            1,2,3,4};
    array double meana1[3], meanb1[4];
    double meanval;
```

```
    mean(transpose(a), meana1);
    printf("mean(traspose(a), meana1) = %f\n", meana1);

    meanval = mean(transpose(b), meanb1);
    printf("mean(transpose(b), meanb1) = %f\n", meanb1);
}
```

Output3

```
mean(traspose(a), meana1) = 2.500000 3.500000 4.500000

mean(transpose(b), meanb1) = 2.333333 3.333333 4.333333 5.333333
```

See Also

median(), **minloc**(), **maxloc**().

References

median

Synopsis
#include <**numeric.h**>
double median(**array double** &*a*, ... /*[**array double** *v*[:]] */);

Syntax
median(a)
median(a, v)

Purpose
Find the median value of all elements of an array and median values of the elements in each row of a two-dimensional array.

Return Value
This function returns the median value of all the elements.

Parameters
a An input array of any dimension.

v An output array which contains the median values of each row.

Description
The function calculates the median value of all elements in an array of any dimension. If the array is a two-dimensional matrix, the function can calculate the median values of each row. The median values of each row are passed out by argument *v*.

Example1
Find the median value of all the elements of arrays.

```
#include <numeric.h>
int main() {
    double a[3] = {1,2,3};
    double b[2][3] = {1,2,3,4,5,6};
    double c[2][3][5];
    c[0][0][0] = 10;
    c[0][0][1] = 20;
    double medianval;

    medianval = median(a);
    printf("median(a) = %f\n", medianval);
    medianval = median(b);
    printf("median(b) = %f\n", medianval);
    medianval = median(c);
    printf("median(c) = %f\n", medianval);
}
```

Output

```
median(a) = 2.000000
median(b) = 3.500000
median(c) = 0.000000
```

Example2

Find the median values of each row of arrays.

```
#include <numeric.h>
int main() {
    double a[2][3] = {1,2,3,
                      4,5,6};
    array double b[3][4] = {1,2,3,4,
                            5,6,7,8,
                            1,2,3,4};
    array double medianv1[2], medianv2[3];

    median(a, medianv1);
    printf("median(a, medianv1) = %f\n", medianv1);
    median(b, medianv2);
    printf("median(b, medianv2) = %f\n", medianv2);
}
```

Output

```
median(a, medianv1) = 2.000000 5.000000

median(b, medianv2) = 2.500000 6.500000 2.500000
```

See Also

mean(), **minv**().

References

minloc

Synopsis
#include <**numeric.h**>
int minloc(**array double** &*a*);

Purpose
Find the index of the element with the minimum value in an array.

Return Value
This function returns the index for the element with the minimum value in an array.

Parameters
a The input array in which the index of the element with the minimum value shall be found.

Description
The function finds the index of the element with the minimum value in the array *a*. The array can be of any dimension.

Example

```
#include <numeric.h>
int main() {
    double a[3] = {1,2,3};
    double b[2][3] = {1,2,3,-10,5,6};
    double c[2][3][5];
    c[0][0][3] = 10;
    c[1][0][3] = -10;
    int loc;

    loc = minloc(a);
    printf("minloc(a) = %d\n", loc);
    loc = minloc(b);
    printf("minloc(b) = %d\n", loc);
    loc = minloc(c);
    printf("minloc(c) = %d\n", loc);
}
```

Output

```
minloc(a) = 0
minloc(b) = 3
minloc(c) = 18
```

See Also
maxloc().

References

minv

Synopsis

#include <**numeric.h**>
array double minv(**array double** *a*[&][&])[:];

Purpose

Find the minimum values of each row in a two-dimensional array.

Return Value

This function returns the array of minimum values.

Parameters

a Input two-dimensional array in which the minimum values of each row will be found.

Description

The function finds the minimum values of each row of a two-dimensional array. The input array can be of any arithmetic data type.

Example

```
#include <numeric.h>
int main() {
    double a[2][3] = {1,2,3,
                      4,5,6};
    array double b[3][4] = {1,2,3,4,
                            5,6,7,8,
                            1,2,3,4};
    array double minv1[2], minv2[3];

    minv1 = minv(a);
    printf("minv(a) = %f\n", minv1);
    minv2 = minv(b);
    printf("minv(b) = %f\n", minv2);
}
```

Output

```
minv(a) = 1.000000 4.000000

minv(b) = 1.000000 5.000000 1.000000
```

See Also

maxloc(), **minloc**(), **median**().

References

norm

Synopsis
#include <**numeric.h**>
double norm(**array double complex** &*a*, **char** * mode);

Purpose
Calculate norm of a vector or matrix.

Return Value
This function returns a norm.

Parameters
a Input, one or two-dimensional array for which a norm is calculated.

mode A character array indicating the type of norm to be calculated.

Description
The norm of a vector or matrix is a scalar that gives some measure of the magnitude of the elements of the vector or matrix. The **norm** function calculates norms of different types for a vector or matrix according to the argument *mode*.

Algorithm
The mode of various different norms for both vectors and matrices are defined below.

Mode	Norm Type	Algorithm
	Norm for Vectors	
`"1"`	1-norm	$\|\|a\|\|_1 = \|a_1\| + \|a_1\| + ... + \|a_n\|$
`"2"`	2-norm	$\|\|a\|\|_2 = (\|a_1\|^2 + \|a_2\|^2 + ... + \|a_n\|^2)^{1/2}$
`"p"`	p-norm	$\|\|a\|\|_p = (\|a_1\|^p + \|a_2\|^p + ... + \|a_n\|^p)^{1/p}$ `"p"` is a floating-point number.
`"i"`	infinity norm	$\|\|a\|\|_\infty = \max_i \|a_i\|$
`"-i"`	negative infinity norm	$\|\|a\|\|_{-\infty} = \min_i \|a_i\|$
	Norm for m-by-n Matrices	
`"1"`	1-norm	$\|\|a\|\|_1 = \max_j \sum_{i=1}^{m} \|a_{ij}\|$
`"2"`	2-norm	$\|\|a\|\|_2 =$ maximum singular value of a
`"i"`	infinity norm	$\|\|a\|\|_\infty = \max_i \sum_{j=1}^{m} \|a_{ij}\|$
`"f"`	Frobenius norm	$\|\|a\|\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} \|a_{ij}\|^2$
`"m"`	norm	$\|\|a\|\| = \max(\mathbf{abs}(A[i][j]))$

Example 1
Calculate the norms of real vectors and matrices.

```
#include <numeric.h>
int main() {
    array double a[2][2] = {2, -4,
                            3, -7};
    array double b[3][2] = {2, -4,
                            3, -7,
                            1, 5};
    array double c[2][3] = {2, -4, 3,
                            -7, 1, 5};
    array double complex d[3] = {2, -4, 3};
    double anorm;

/* a is nxn */
    anorm = norm(a, "1");
    printf("1-norm of a = %f\n", anorm);
    anorm = norm(a, "2");
    printf("2-norm of a = %f\n", anorm);
    anorm = norm(a, "i");
    printf("infinity-norm of a = %f\n", anorm);
    anorm = norm(a, "f");
    printf("Frobenius-norm of a = %f\n", anorm);
    anorm = norm(a, "m");
    printf("max(abs(a)) = %f\n\n", anorm);

/* a is nxm */
    anorm = norm(b, "1");
    printf("1-norm of b = %f\n", anorm);
    anorm = norm(b, "2");
    printf("2-norm of b = %f\n", anorm);
    anorm = norm(b, "i");
    printf("infinity-norm of b = %f\n", anorm);
    anorm = norm(b, "f");
    printf("Frobenius-norm of b = %f\n", anorm);
    anorm = norm(b, "m");
    printf("max(abs(a)) = %f\n\n", anorm);

/* a is mxn */
    anorm = norm(c, "1");
    printf("1-norm of c = %f\n", anorm);
    anorm = norm(c, "2");
    printf("2-norm of c = %f\n", anorm);
    anorm = norm(c, "i");
    printf("infinity-norm of c = %f\n", anorm);
    anorm = norm(c, "f");
    printf("Frobenius-norm of c = %f\n", anorm);
    anorm = norm(c, "m");
    printf("max(abs(a)) = %f\n\n", anorm);

/* a is 1-dim */
    anorm = norm(d, "1");
    printf("1-norm of d = %f\n", anorm);
    anorm = norm(d, "2");
    printf("2-norm of d = %f\n", anorm);
    anorm = norm(d, "2.1");
    printf("2.1-norm of d = %f\n", anorm);
    anorm = norm(d, "i");
    printf("infinity-norm of d = %f\n", anorm);
    anorm = norm(d, "-i");
    printf("negative infinity-norm of d = %f\n", anorm);
```

```
}
```

Output1

```
1-norm of a = 11.000000
2-norm of a = 8.828855
infinity-norm of a = 10.000000
Frobenius-norm of a = 8.831761
max(abs(a)) = 7.000000

1-norm of b = 16.000000
2-norm of b = 8.671495
infinity-norm of b = 10.000000
Frobenius-norm of b = 10.198039
max(abs(a)) = 7.000000

1-norm of c = 9.000000
2-norm of c = 9.846035
infinity-norm of c = 13.000000
Frobenius-norm of c = 10.198039
max(abs(a)) = 7.000000

1-norm of d = 9.000000
2-norm of d = 5.385165
2.1-norm of d = 5.263628
infinity-norm of d = 4.000000
negative infinity-norm of d = 2.000000
```

Example 2
Calculate the norms of complex vectors and matrices.

```
#include <numeric.h>
int main() {
    array double complex z[2][2] = {2, -4,
                                    3, -7};
    array complex z1[2][2]        = {2, -4,
                                    3, -7};
    array double complex z2[2][2] = {2, -4,
                                    3, complex(1,-7)};
    array double complex z3[4] = {2, complex(-4,1),3};
    double anorm;

    anorm = norm(z, "1");
    printf("1-norm of a = %f\n", anorm);
    anorm = norm(z, "i");
    printf("infinity-norm of a = %f\n", anorm);
    anorm = norm(z, "f");
    printf("Frobenius-norm of a = %f\n", anorm);
    anorm = norm(z, "m");
    printf("max(abs(a)) = %f\n\n", anorm);

    anorm = norm(z1, "1");
    printf("1-norm of a = %f\n", anorm);
    anorm = norm(z1, "i");
    printf("infinity-norm of a = %f\n", anorm);
    anorm = norm(z1, "f");
    printf("Frobenius-norm of a = %f\n", anorm);
    anorm = norm(z1, "m");
    printf("max(abs(a)) = %f\n\n", anorm);
```

```
    anorm = norm(z2, "1");
    printf("1-norm of z2 = %f\n", anorm);
    anorm = norm(z2, "i");
    printf("infinity-norm of z2 = %f\n", anorm);
    anorm = norm(z2, "f");
    printf("Frobenius-norm of z2 = %f\n", anorm);
    anorm = norm(z2, "m");
    printf("max(abs(z2)) = %f\n\n", anorm);

    anorm = norm(z3, "1");
    printf("1-norm of d = %f\n", anorm);
    anorm = norm(z3, "2");
    printf("2-norm of d = %f\n", anorm);
    anorm = norm(z3, "2.1");
    printf("2.1-norm of d = %f\n", anorm);
    anorm = norm(z3, "i");
    printf("infinity-norm of d = %f\n", anorm);
    anorm = norm(z3, "-i");
    printf("negative infinity-norm of d = %f\n", anorm);

}
```

Output2

```
1-norm of a = 11.000000
infinity-norm of a = 10.000000
Frobenius-norm of a = 8.831761
max(abs(a)) = 7.000000

1-norm of a = 11.000000
infinity-norm of a = 10.000000
Frobenius-norm of a = 8.831761
max(abs(a)) = 7.000000

1-norm of z2 = 11.071068
infinity-norm of z2 = 10.071068
Frobenius-norm of z2 = 8.888194
max(abs(z2)) = 7.071068

1-norm of d = 9.123106
2-norm of d = 5.477226
2.1-norm of d = 5.355310
infinity-norm of d = 4.123106
negative infinity-norm of d = 0.000000
```

See Also

condnum(), **rcondnum**(), **svd**(), **maxloc**(), **maxv**(), **minloc**(), **minv**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

nullspace

Synopsis
#include <numeric.h>
int nullspace(array double complex *null*[&][&], **array double complex** *a*[&][&]);

Purpose
Calculate null space of matrix *a*.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
a Input matrix.

null Output array which contains the null space of matrix *a*.

Description
This function calculates an orthonormal bases for null space of matrix *a*. The null space *s* meets the following condition

$$s^t * s = I$$

and

$$a * s = 0$$

Algorithm
The computation of null space of matrix *a* is based on the singular value decomposition of matrix *a*. The SVD is formulated as

$$A = USV^T$$

Where S is an m-by-n matrix which is zero except for its min(m,n) diagonal elements, U is an m-by-m orthogonal matrix, and V is an n-by-n orthogonal matrix. The null space can be obtained by

$$orth(i, j) = V^T(i, j),$$

with i=0, 1, 2, ..., n-1, and j=r, r+1, ... , n-1, where r is the number of non-zero singular values with tolerance $tol = \max(m, n) \times \max(S) \times$ `DBL_EPSILON`.

Example
Calculate null space of a singular matrix with rank(a) == 2.

```
#include <numeric.h>
#define M 3
#define N 3
int main() {
    /* singular matrix rank(a) == 2 */
    array double a[M][N] = {1, 2, 3,
                            4, 5, 6,
                            7, 8, 9};
    int r = rank(a);
    array double null[M][N-r];

    nullspace(null, a);
    printf("rank(a) = %d\n", r);
    printf("null from nullspace(null, a) = \n%f\n", null);
    printf("transpose(null)*null = \n%f\n", transpose(null)*null);
}
```

Output

```
rank(a) = 2
null from nullspace(null, a) =
-0.408248
0.816497
-0.408248

transpose(null)*null =
1.000000
```

Example2
Calculate null space of a singular matrix with rank(a) == 1.

```
#include <numeric.h>
#define M 3
#define N 3
int main() {
    /* singular matrix rank(a) == 1 */
    array double a[M][N] = {1, 2, 3,
                            1, 2, 3,
                            2, 4, 6};
    int r = rank(a);
    array double null[M][N-r];

    nullspace(null, a);
    printf("rank(a) = %d\n", r);
    printf("null from nullspace(null, a) = \n%f\n", null);
    printf("transpose(null)*null = \n%f\n", transpose(null)*null);
}
```

Output2

```
rank(a) = 1
null from nullspace(null, a) =
-0.872872 0.408248
-0.218218 -0.816497
0.436436 0.408248

transpose(null)*null =
1.000000 -0.000000
-0.000000 1.000000
```

Example3
Calculate null space of a non-singular matrix with rank(a) == 3.

```
#include <numeric.h>
#define M 3
#define N 3
int main() {
    /* non-singular matrix rank(a) == 3 */
    array double a[M][N] = {1, 2, 3,
                            5, 5, 6,
                            7, 8, 9};
    int r = rank(a);
    if(r == 3) {
      printf("rank(a) is 3, empty null space\n");
      return 0;
    }
    else  {
      array double null[M][N-r];
      nullspace(null, a);
      printf("rank(a) = %d\n", r);
      printf("null from nullspace(null, a) = \n%f\n", null);
      printf("transpose(null)*null = \n%f\n", transpose(null)*null);
    }
}
```

Output3

```
rank(a) is 3, empty null space
```

See Also

svd(), **rank**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

oderk

Synopsis
#include <**numeric.h**>
int oderk(**double** (***func*)(**double**, **double**[], **double**[], **void***),
double *t0*, **double** *tf*, **double** *y0*[:], **void** **param*,
double *t*[:], **double** **y*, ... /* [**double** *eps*] */);

Syntax
oderk(*func*, *t0*, *tf*, *y0*, `param`, *t*, *y*)
oderk(*func*, *t0*, *tf*, *y0*, `param`, *t*, *y*, *eps*)

Purpose
Numerical solution for ordinary differential equations (ODE) using Runge-Kutta method.

Return Value
If successful, this function returns the number of points calculated in the interval between *t0* and *tf*. Otherwise, it returns -1.

Parameters
func A pointer to function, the function shall be first-order differential equation.

t0 A start point of *t*.

tf A final point of *t*.

y0 A vector which contains the initial values of ODE functions.

param An option parameter for passing information to ODE functions.

t A vector which contains the *t* value in the interval

y The address for a one or two-dimensional array which contains the solution of the ODE function. For a two-dimensional array, each row of *y* corresponds to each derivative function and each column corresponds to the interval value t_i between *t0* and *xf*.

eps The user specified tolerance. If the user does not specify this option, the value of 10^{-8} is used by default.

Description
The derivative functions *func* is specified as a pointer to a function which shall be first-order differential equation. The end-points *t0* and *tf* and initial value of differential equations *y0* shall be array of double type. The returned **int** value is the effective number of points calculated in the interval between *t0* and *tf*. Vector

t contains the values between *t0* and *tf* in which the ODE function is solved and the results are stored in a one-dimensional array *y* for an ordinary differential equation or in a two-dimensional array for a system of ordinary differential equations. The argument `param` is used to pass information from an application program to the ODE functions. If the optional argument *eps* is passed, the algorithm uses the user specified tolerance to decide when to stop the iterations. Otherwise, the value of 10^{-8} is used by default.

Algorithm

Fifth-order Runge-Kutta with adaptive stepsize control.

$$y_{n+1} = y_n + c_1k_1 + c_2k_2 + c_3k_3 + c_4k_4 + c_5k_5 + c_6k_6 + O(h^6)$$

Where

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + a_2h, y_n + b_{21}k_1) \\ &\vdots \quad \vdots \quad \vdots \\ k_6 &= hf(t_n + a_6h, y_n + b_{61}k_1 + \cdots + b_{65}k_5) \end{aligned}$$

The particular values of the various constants are obtained by Cash and Karp, and given in the table below.

Cash-Karp Parameters for Embedded Runga-Kutta Method

i	a_i	b_i					c_i
1							$\frac{37}{378}$
2	$\frac{1}{5}$	$\frac{1}{5}$					0
3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				$\frac{250}{621}$
4	$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			$\frac{125}{594}$
5	1	$-\frac{11}{54}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		0
6	$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	$\frac{512}{1771}$
j =		1	2	3	4	5	

Example 1

This example shows how to solve the first-order derivative function using function **oderk**(). We want to solve ODE function

$$\frac{dy}{dt} = cos(t)$$

with $t_0 = 0, t_f = 2\pi, y_0 = 0$ and no more than 50 points.

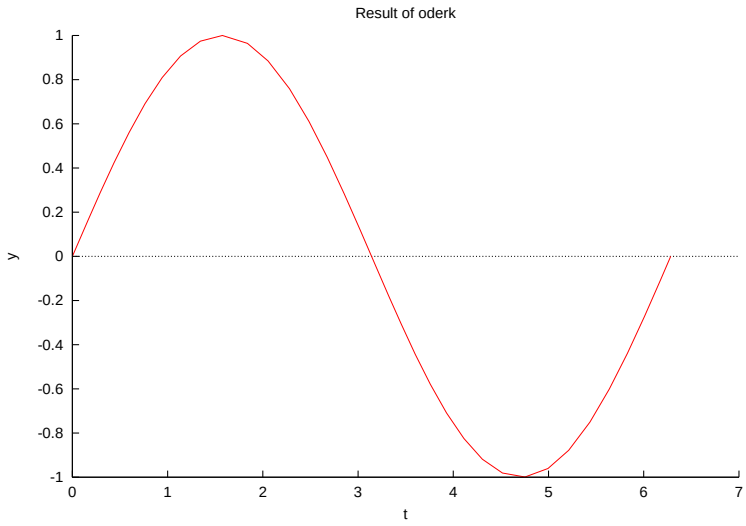
```
#include <stdio.h>
#include <numeric.h>

#define NVAR 1
#define POINTS 50

int main() {
```

```
    void func(double t, double y[], double dydt[], void *param);
    double t0=0, tf=2*M_PI, y0[NVAR]={0};
    double t[POINTS], y[POINTS];
    int i, points;

/* return -1: divergent
   return >1: ok, num of points
*/
    points = oderk(func, t0, tf, y0, NULL, t, y);
    printf("points = %d\n", points);
    if(points > 0) {
      printf("\n%8s %18s %15s\n","t","oderk()","sin(t)");
      for (i=0; i<points; i++)
         printf("%10.4f %14.6f %14.6f\n", t[i],y[i],sin(t[i]));
      plotxy(t, y, "result of oderk", "t", "y");
    }
    else
      printf("oderk() failed\n");
}

void func(double t, double y[], double dydt[], void *param) {
    /* y' = cos(t) */
    dydt[0] = cos(t);
}
```

Output

```
points = 35

       t            oderk()          sin(t)
    0.0000       0.000000       0.000000
    0.1647       0.163958       0.163958
    0.2930       0.288791       0.288791
    0.4373       0.423485       0.423485
    0.5945       0.560114       0.560114
    0.7633       0.691328       0.691328
    0.9435       0.809628       0.809628
    1.1361       0.907004       0.907004
    1.3441       0.974410       0.974410
    1.5755       0.999989       0.999989
    1.8381       0.964487       0.964487
    2.0552       0.884972       0.884972
    2.2800       0.758904       0.758904
    2.4850       0.610393       0.610393
    2.6762       0.448786       0.448786
    2.8559       0.281835       0.281835
    3.0246       0.116702       0.116702
    3.1814      -0.039813      -0.039813
    3.3231      -0.180529      -0.180529
    3.4539      -0.307275      -0.307275
    3.6002      -0.442691      -0.442691
    3.7591      -0.579004      -0.579004
    3.9295      -0.708868      -0.708868
    4.1113      -0.824733      -0.824733
    4.3058      -0.918487      -0.918487
    4.5164      -0.980855      -0.980855
    4.7526      -0.999192      -0.999192
```

```
    4.9945      -0.960467      -0.960467
    5.2116      -0.877948      -0.877948
    5.4347      -0.750252      -0.750252
    5.6388      -0.600703      -0.600703
    5.8292      -0.438549      -0.438549
    6.0082      -0.271508      -0.271508
    6.1763      -0.106713      -0.106713
    6.2832       0.000000      -0.000000
```



Example 2

This example shows how to specify the tolerance. We want to solve ODE function

$$\frac{dy}{dt} = cos(t)$$

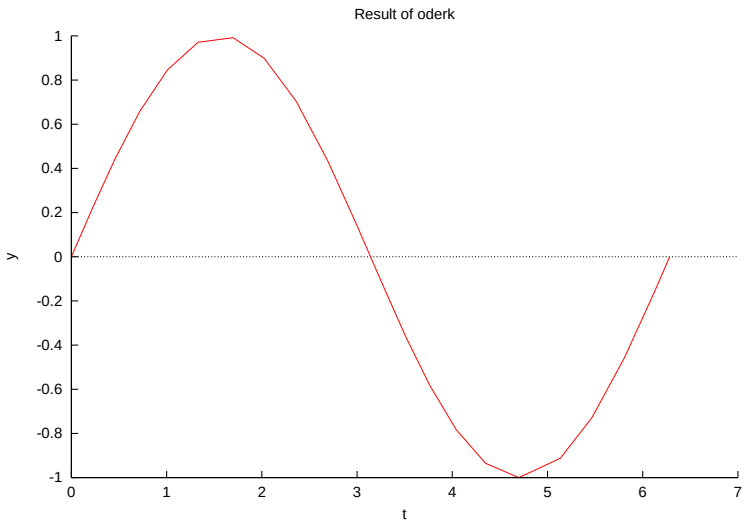
with $t_0 = 0, t_f = 2\pi, y_0 = 0$ and no more than 50 points. The numerical tolerance is FLT_EPSILON instead of 10^{-8} by default.

```
#include <stdio.h>
#include <float.h>
#include <numeric.h>

#define NVAR 1
#define POINTS 50

int main() {
    void func(double t, double y[], double dydt[], void *param);
    double t0=0, tf=2*M_PI, y0[NVAR]={0};
    double t[POINTS], y[POINTS];
    double tol;
    int i, points;

    tol = FLT_EPSILON; /* default tolerance is 1e-6 */
    //points = oderk(t, y, NULL, func, t0, tf, y0, tol);
    points = oderk(func, t0, tf, y0, NULL, t, y);
    printf("points = %d\n", points);
    printf("\n%8s %18s %15s\n","t","oderk()","sin(t)");
    for (i=0;i<points;i++)
```

```
      printf("%10.4f %14.6f %14.6f\n", t[i],y[i],sin(t[i]));
    plotxy(t, y, "result of oderk", "t", "y");
}

void func(double t, double y[], double dydt[], void *param) {
    /* y' = cos(t) */
    dydt[0] = cos(t);
}
```

Output

```
points = 22

         t             oderk()           sin(t)
    0.0000        0.000000        0.000000
    0.2319        0.229803        0.229803
    0.4576        0.441772        0.441772
    0.7181        0.657918        0.657918
    1.0080        0.845762        0.845762
    1.3295        0.971026        0.971026
    1.6979        0.991935        0.991935
    2.0250        0.898604        0.898604
    2.3583        0.705591        0.705591
    2.6974        0.429731        0.429731
    3.0073        0.133918        0.133918
    3.2598       -0.117907       -0.117907
    3.5128       -0.362780       -0.362781
    3.7617       -0.581135       -0.581135
    4.0411       -0.783034       -0.783034
    4.3504       -0.935198       -0.935198
    4.6982       -0.999900       -0.999900
    5.1342       -0.912350       -0.912350
    5.4667       -0.728744       -0.728744
    5.8095       -0.456146       -0.456146
    6.1219       -0.160581       -0.160581
    6.2832       -0.000000       -0.000000
```



Example 3

This example shows how to solve the Van der Pol equation

$$\frac{d^2u}{dt^2} - \mu(1-u^2)\frac{du}{dt} + u = 0$$

where $\mu = 2, t_0 = 1, t_f = 30, u(t_0) = 1$, and $u'(t_0) = 0$;
The Van der Pol equation is reformulated as a set of first-order differential equations first.
Let

$$\begin{aligned} y_0 &= u \\ y_1 &= \frac{du}{dt} \end{aligned}$$

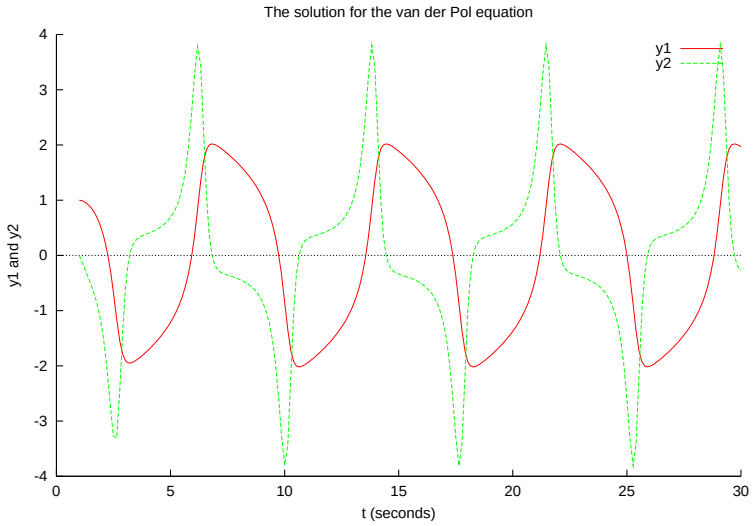
then

$$\begin{aligned} \frac{dy_0}{dt} &= \frac{du}{dt} \\ \frac{dy_1}{dt} &= \frac{d^2u}{dt^2} = \mu(1-u^2)\frac{du}{dt} + u = \mu(1-y_0^2)y_1 - y_0 \end{aligned}$$

Use **oderk**() to solve this system of ordinary differential equations. The parameter `param` is used to pass μ from the function main() to the ODE function `func()`.

```
#include <chplot.h>
#include <numeric.h>

#define NVAR 2
#define POINTS 256
void func(double t, double y[], double dydt[], void *param) {
    double mu;
    mu = *(double*)param;
    dydt[0] = y[1];
    dydt[1]=mu*(1-y[0]*y[0])*y[1] - y[0];
}

int main() {
    double t0=1, tf=30, y0[NVAR] = {1, 0};
    double t[POINTS], y[NVAR][POINTS];
    double mu = 2;

    oderk(func, t0, tf, y0, &mu, t, y);
    plotxy(t, y, "The solution for the van der Pol equation", "t (seconds)", "y1 and y2");
}
```

Output



Example 4

Function **oderk**() returns the number of points calculated in the interval *t0* and *tf*. Generally the user should guess how many points it would produce and define the size of **array** *t* and *y* a little bit bigger than the guessed number of points. Function **oderk**() will automatically pad the leftover space in *t* and *y* with results at *tf*.

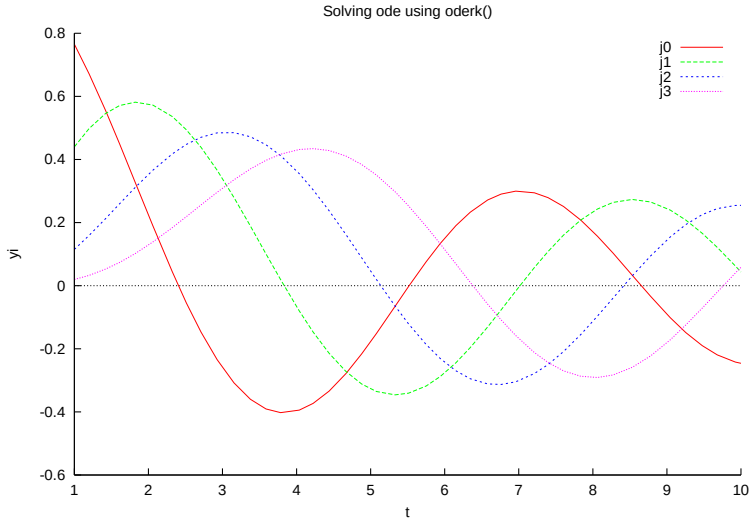
```
#include <numeric.h>
#include <stdio.h>
#include <chplot.h>

#define NVAR 4
#define POINTS 50

int main() {
    double t0=1, tf=10, y0[NVAR];
    void func(double t, double y[], double dydt[], void *param);
    double t[POINTS], y[NVAR][POINTS];
    int i, points;
    class CPlot plot;

    y0[0]=j0(t0);
    y0[1]=j1(t0);
    y0[2]=jn(2,t0);
    y0[3]=jn(3,t0);

    points = oderk(func, t0, tf, y0, NULL, t, y);
    printf("points = %d\n", points);
    printf("\n%8s %18s %15s\n","t","integral","bessj(2,t)");
    for (i=0; i<points; i++)
       printf("%10.4f %14.6f %14.6f\n", t[i],y[2][i],jn(2,t[i]));

    printf("\nThe leftover space in t and y are padded with results at tf\n");
    for (i=points; i<POINTS; i++)
       printf("%10.4f %14.6f %14.6f\n", t[i],y[2][i],jn(2,t[i]));

    plotxy(t, y, "solving ode using oderk()", "t", "yi", &plot);
    plot.legend("j0", 0);
    plot.legend("j1", 1);
```

```
    plot.legend("j2", 2);
    plot.legend("j3", 3);
    plot.plotting();
}

void func(double t, double y[], double dydt[], void *param) {
    dydt[0] = -y[1];
    dydt[1]=y[0]-(1.0/t)*y[1];
    dydt[2]=y[1]-(2.0/t)*y[2];
    dydt[3]=y[2]-(3.0/t)*y[3];
}
```

Output

```
points = 43

       t           integral      bessj(2,t)
    1.0000       0.114903       0.114903
    1.1995       0.159225       0.159225
    1.4208       0.212473       0.212473
    1.6107       0.259628       0.259628
    1.8230       0.311687       0.311687
    2.0595       0.365944       0.365944
    2.3217       0.417776       0.417776
    2.5117       0.447689       0.447689
    2.7100       0.470487       0.470487
    2.9266       0.484255       0.484255
    3.1546       0.485094       0.485094
    3.3763       0.471942       0.471942
    3.5876       0.446661       0.446661
    3.7842       0.412467       0.412467
    4.0366       0.354910       0.354910
    4.2242       0.303546       0.303546
    4.4402       0.237267       0.237267
    4.6658       0.162285       0.162285
    4.8797       0.088340       0.088340
    5.0780       0.019679       0.019679
    5.3301      -0.064486      -0.064486
    5.5121      -0.120926      -0.120926
    5.7491      -0.186326      -0.186326
    5.9414      -0.230973      -0.230973
    6.1517      -0.269627      -0.269627
    6.3458      -0.294896      -0.294896
    6.5806      -0.311270      -0.311270
    6.7618      -0.313089      -0.313089
    6.9592      -0.304516      -0.304516
    7.2117      -0.278417      -0.278417
    7.3951      -0.249841      -0.249841
    7.6042      -0.208797      -0.208797
    7.8362      -0.154879      -0.154879
    8.0631      -0.096271      -0.096271
    8.2728      -0.039407      -0.039407
    8.5323       0.030949       0.030949
    8.7792       0.094212       0.094212
    9.0451       0.154300       0.154300
    9.2558       0.193600       0.193600
    9.4769       0.225200       0.225200
    9.6799       0.244472       0.244472
```

```
    9.9231       0.254481       0.254481
   10.0000       0.254630       0.254630

The leftover space in t and y are padded with results at tf
   10.0000       0.254630       0.254630
   10.0000       0.254630       0.254630
   10.0000       0.254630       0.254630
   10.0000       0.254630       0.254630
   10.0000       0.254630       0.254630
   10.0000       0.254630       0.254630
   10.0000       0.254630       0.254630
```



See Also

derivative(), **integral1**(), **integral2**(), **integral3**(), **integration2**(), **integration3**().

References

Cash, J. R., and Karp, A. H., *ACM Transactions on Mathematical Software*, 1990. vol. 16, pp. 201-222.

oderungekutta

Synopsis
#include <**numeric.h**>
int oderungekutta(**double** *t*[&], **double** &*y*, **void** **param*,
double (**func*)(**double**, **double**[], **double**[], **void***),
double *t0*, **double** *tf*, **double** *y0*[&], ... /* [**double** *eps*] */);

Syntax
oderungekutta(*t*, *y*, `param`, *func*, *t0*, *tf*, *y0*)
oderungekutta(*t*, *y*, `param`, *func*, *t0*, *tf*, *y0*, *eps*)

Purpose
Numerical solution for ordinary differential equations (ODE) using Runge-Kutta method. This function is obsolete. Use function **oderk**().

Return Value
If successful, this function returns the number of points calculated in the interval between *t0* and *tf*. Otherwise, it returns -1.

Parameters
t A vector which contains the *t* value in the interval between *t0* and *xf*.

y A one or two-dimensional array which contains the solution of the ODE function. For a two-dimensional array, each row of *y* corresponds to each derivative function and each column corresponds to the interval value t_i

param An option parameter for passing information to ODE functions.

func A pointer to function, the function shall be first-order differential equation.

t0 A start point of *t*.

tf A final point of *t*.

y0 A vector which contains the initial values of ODE functions.

eps The user specified tolerance. If the user does not specify this option, the value of 10^{-8} is used by default.

Description
The derivative functions *func* is specified as a pointer to a function which shall be first-order differential

equation. The end-points *t0* and *tf* and initial value of differential equations *y0* can be any **real** data type. Conversion of the data to double type is performed internally. The returned **int** value is the effective number of points calculated in the interval between *t0* and *tf*. Vector *t* contains the values between *t0* and *tf* in which the ODE function is solved and the results are stored in a one-dimensional array *y* for an ordinary differential equation or in a two-dimensional array for a system of ordinary differential equations. The argument `param` is used to pass information from an application program to the ODE functions. If the optional argument *eps* is passed, the algorithm uses the user specified tolerance to decide when to stop the iterations. Otherwise, the value of 10^{-8} is used by default.

Algorithm

Fifth-order Runge-Kutta with adaptive stepsize control.

$$y_{n+1} = y_n + c_1 k_1 + c_2 k_2 + c_3 k_3 + c_4 k_4 + c_5 k_5 + c_6 k_6 + O(h^6)$$

Where

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + a_2 h, y_n + b_{21} k_1) \\ &\vdots \quad \vdots \quad \vdots \\ k_6 &= hf(t_n + a_6 h, y_n + b_{61} k_1 + \cdots + b_{65} k_5) \end{aligned}$$

The particular values of the various constants are obtained by Cash and Karp, and given in the table below.

Cash-Karp Parameters for Embedded Runga-Kutta Method							
i	a_i	b_i					c_i
1							$\frac{37}{378}$
2	$\frac{1}{5}$	$\frac{1}{5}$					0
3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				$\frac{250}{621}$
4	$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			$\frac{125}{594}$
5	1	$-\frac{11}{54}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		0
6	$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	$\frac{512}{1771}$
j =		1	2	3	4	5	

See Also

oderk().

References

Cash, J. R., and Karp, A. H., *ACM Transactions on Mathematical Software*, 1990. vol. 16, pp. 201-222.

odesolve

Synopsis
#include <**numeric.h**>
int odesolve(**double** *t*[&], **double** &*y*, **double** (**f*unc*)(**double**, **double**[], double[]), **double** *t0*, **double** *tf*, **double** *y0*[&],
... /* [**double** *eps*] */);

Syntax
odesolve(*t*, *y*, *func*, *t0*, *tf*, *y0*)
odesolve(*t*, *y*, *func*, *t0*, *tf*, *y0*, *eps*)

Purpose
Numerical solution for ordinary differential equations (ODE). This function is obsolete. Use function **oderk**().

Return Value
If successful, this function returns the number of points calculated in the interval between *t0* and *tf*. Otherwise, it returns -1.

Parameters
t A vector which contains the *t* value in the interval between *t0* and *xf*.

y A one or two-dimensional array which contains the solution of the ODE function. For a two-dimensional array, each row of *y* corresponds to each derivative function and each column corresponds to the interval value t_i

func A pointer to function, the function shall be first-order differential equation.

t0 A start point of *t*.

tf A final point of *t*.

y0 A vector which contains the initial values of ODE functions.

eps The user specified tolerance. If the user does not specify this option, the value of 10^{-8} is used by default.

Description
The derivative functions *func* is specified as a pointer to a function which shall be first-order differential equation. The end-points *t0* and *tf* and initial value of differential equations *y0* can be any **real** data type. Conversion of the data to double type is performed internally. The returned **int** value is the effective number of points calculated in the interval between *t0* and *tf*. Vector *t* contains the values between *t0* and *tf* in

which the ODE function is solved and the results are stored in a one-dimensional array *y* for an ordinary differential equation or in a two-dimensional array for a system of ordinary differential equations. If the optional argument *eps* is passed, the algorithm uses the user specified tolerance to decide when to stop the iterations. Otherwise, the value of 10^{-8} is used by default.

Algorithm
Fifth-order Runge-Kutta with adaptive stepsize control.

$$y_{n+1} = y_n + c_1 k_1 + c_2 k_2 + c_3 k_3 + c_4 k_4 + c_5 k_5 + c_6 k_6 + O(h^6)$$

Where

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + a_2 h, y_n + b_{21} k_1) \\ &\vdots \quad \vdots \quad \vdots \\ k_6 &= hf(t_n + a_6 h, y_n + b_{61} k_1 + \cdots + b_{65} k_5) \end{aligned}$$

The particular values of the various constants are obtained by Cash and Karp, and given in the table below.

Cash-Karp Parameters for Embedded Runga-Kutta Method							
i	a_i	b_i					c_i
1							$\frac{37}{378}$
2	$\frac{1}{5}$	$\frac{1}{5}$					0
3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				$\frac{250}{621}$
4	$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			$\frac{125}{594}$
5	1	$-\frac{11}{54}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		0
6	$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	$\frac{512}{1771}$
j =		1	2	3	4	5	

see also
oderk().
References
Cash, J. R., and Karp, A. H., *ACM Transactions on Mathematical Software*, 1990. vol. 16, pp. 201-222.

orthonormalbase

Synopsis
#include <**numeric.h**>
int orthonormalbase(**array double complex** *orth*[&][&], **array double complex** *a*[&][&]);

Purpose
Find orthonormal bases of a matrix.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
a The input matrix.

orth An output matrix of orthonormal bases for the matrix *a*.

Description
This function calculates the orthonormal bases for a matrix. The columns of *orth* are orthonormal, and have the same space as columns of *a*. The number of columns of *orth* is the rank of *a*.

Algorithm
The computation of orthonormal bases for matrix *a* is based on the singular value decomposition(SVD) of matrix *a*. The SVD is formulated as

$$A = USV^T$$

Where S is an m-by-n matrix which is zero except for its min(m,n) diagonal elements, U is an m-by-m orthogonal matrix, and V is an n-by-n orthogonal matrix. The orthonormal bases can be obtained by

$$orth(i,j) = U(i,j),$$

with i=0, 1, 2, ... n-1, and j=0, 1, 2, ... r-1, where r is the number of non-zero singular values with tolerance $tol = \max(m,n) \times \max(S) \times$ `DBL_EPSILON`.

Example1
Calculate the orthonormal bases of a singular 3-by-3 matrix with rank(a) == 2.

```
#include <numeric.h>
#define M 3
#define N 3
int main() {
    /* singular matrix rank(a) == 2 */
    array double a[M][N] = {1, 2, 3,
                            4, 5, 6,
                            7, 8, 9};
    int r = rank(a);
```

```
    array double orth[M][r];

    orthonormalbase(orth, a);
    printf("rank(a) = %d\n", r);
    printf("orth from orthonormalbase(orth, a) = \n%f\n", orth);
    printf("transpose(orth)*orth = \n%f\n", transpose(orth)*orth);
}
```

Output1

```
rank(a) = 2
orth from orthonormalbase(orth, a) =
-0.214837 0.887231
-0.520587 0.249644
-0.826338 -0.387943

transpose(orth)*orth =
1.000000 -0.000000
-0.000000 1.000000
```

Example2
Calculate the orthonormal bases of a singular 3-by-3 matrix with rank(a) == 1.

```
#include <numeric.h>
#define M 3
#define N 3
int main() {
    /* singular matrix rank(a) == 1 */
    array double a[M][N] = {1, 2, 3,
                            1, 2, 3,
                            2, 4, 6};
    int r = rank(a);
    array double orth[M][r];

    orthonormalbase(orth, a);
    printf("rank(a) = %d\n", r);
    printf("orth from orthonormalbase(orth, a) = \n%f\n", orth);
    printf("transpose(orth)*orth = \n%f\n", transpose(orth)*orth);
}
```

Output2

```
rank(a) = 1
orth from orthonormalbase(orth, a) =
-0.408248
-0.408248
-0.816497

transpose(orth)*orth =
1.000000
```

Example3
Calculate the orthonormal bases of a singular 3-by-3 matrix with rank(a) == 3.

```
#include <numeric.h>
#define M 3
#define N 3
int main() {
    /* singular matrix rank(a) == 3 */
```

```
    array double a[M][N] = {1, 2, 3,
                            5, 5, 6,
                            7, 8, 9};
    int r = rank(a);
    array double orth[M][r];

    orthonormalbase(orth, a);
    printf("rank(a) = %d\n", r);
    printf("orth from orthonormalbase(orth, a) = \n%f\n", orth);
    printf("transpose(orth)*orth = \n%f\n", transpose(orth)*orth);
}
```

Output3

```
rank(a) = 3
orth from orthonormalbase(orth, a) =
-0.210138 0.955081 -0.208954
-0.541507 -0.291649 -0.788486
-0.814010 -0.052541 0.578470

transpose(orth)*orth =
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
```

See Also

nullspace(), **svd**(), **rank**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

pinverse

Synopsis
#include <numeric.h>
array double pinverse(array double *a*[:][:])[:][:];

Purpose
Calculate MoorePenrose pseudo inverse of a matrix.

Return Value
This function returns the pseudo inverse of a matrix.

Parameters
a An input matrix.

Description
The Moore_Penrose pseudo inverse matrix *b* of matrix *a* has to meet the following four conditions:

$$a * b * a = a,$$
$$b * a * b = b,$$
$$a * b \text{ is Hermitian}$$
$$b * a \text{ is Hermitian}$$

Algorithm
The algorithm of this function is based on the function **svd**(). Any singular value less than DBL_EPSILON is treated as zero.

Example1
Calculate the Moore-Penrose pseudo inverse of a 3-by-3 singular matrix with rank(a) == 2.

```
#include <numeric.h>
#define M 3
#define N 3
/* pseudoinverse */
int main() {
    /* singular matrix rank(a) == 2 */
    array double a[M][N] = {1, 2, 3,
                            4, 5, 6,
                            7, 8, 9};
    int r;
    array double pinv[N][M];

    r = rank(a);
    pinv = pinverse(a);
    printf("rank(a) = %d\n", r);
```

```
    printf("pinverse(a) = \n%f\n", pinv);
    printf("a*pinverse(a)*a = \n%f\n", a*pinv*a);
    printf("pinverse(a)*a*pinverse(a) = \n%f\n", pinv*a*pinv);
}
```

Output1

```
rank(a) = 2
pinverse(a) =
-0.638889 -0.166667 0.305556
-0.055556 0.000000 0.055556
0.527778 0.166667 -0.194444

a*pinverse(a)*a =
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000

pinverse(a)*a*pinverse(a) =
-0.638889 -0.166667 0.305556
-0.055556 0.000000 0.055556
0.527778 0.166667 -0.194444
```

Example2
Calculate the Moore-Penrose pseudo inverse of a 3-by-3 singular matrix with rank(a) == 1.

```
#include <numeric.h>
#define M 3
#define N 3
/* pseudoinverse */
int main() {
    /* singular matrix rank(a) == 1 */
    array double a[M][N] = {1, 2, 3,
                            1, 2, 3,
                            2, 4, 6};
    int r;
    array double pinv[N][M];

    r = rank(a);
    pinv = pinverse(a);
    printf("rank(a) = %d\n", r);
    printf("pinverse(a) = \n%f\n", pinv);
    printf("a*pinverse(a)*a = \n%f\n", a*pinv*a);
    printf("pinverse(a)*a*pinverse(a) = \n%f\n", pinv*a*pinv);
}
```

Output2

```
rank(a) = 1
pinverse(a) =
0.011905 0.011905 0.023810
0.023810 0.023810 0.047619
0.035714 0.035714 0.071429

a*pinverse(a)*a =
1.000000 2.000000 3.000000
1.000000 2.000000 3.000000
2.000000 4.000000 6.000000
```

```
pinverse(a)*a*pinverse(a) =
0.011905 0.011905 0.023810
0.023810 0.023810 0.047619
0.035714 0.035714 0.071429
```

Example3
Calculate the Moore-Penrose pseudo inverse of a 3-by-3 non-singular matrix with rank(a) == 3.

```
#include <numeric.h>
#define M 3
#define N 3
/* pseudoinverse */
int main() {
    /* non-singular matrix rank(a) == 3 */
    array double a[M][N] = {1, 2, 3,
                            5, 5, 6,
                            7, 8, 9};
    int r;
    array double pinv[N][M];

    r = rank(a);
    pinv = pinverse(a);
    printf("rank(a) = %d\n", r);
    printf("pinverse(a) = \n%f\n", pinv);
    printf("a*pinverse(a)*a = \n%f\n", a*pinv*a);
    printf("pinverse(a)*a*pinverse(a) = \n%f\n", pinv*a*pinv);
}
```

Output3

```
rank(a) = 3
pinverse(a) =
-0.500000 1.000000 -0.500000
-0.500000 -2.000000 1.500000
0.833333 1.000000 -0.833333

a*pinverse(a)*a =
1.000000 2.000000 3.000000
5.000000 5.000000 6.000000
7.000000 8.000000 9.000000

pinverse(a)*a*pinverse(a) =
-0.500000 1.000000 -0.500000
-0.500000 -2.000000 1.500000
0.833333 1.000000 -0.833333
```

Example4
Calculate the Moore-Penrose pseudo inverse of a 2-by-3 matrix.

```
#include <numeric.h>
#define M 2
#define N 3
/* pseudoinverse */
int main() {
     array double a[M][N] = {7, 8, 1,
                             3, 6, 4}; /* m-by-n matrix */
    int r;
```

```
    array double pinv[N][M];

    r = rank(a);
    pinv = pinverse(a);
    printf("rank(a) = %d\n", r);
    printf("pinverse(a) = \n%f\n", pinv);
    printf("a*pinverse(a)*a = \n%f\n", a*pinv*a);
    printf("pinverse(a)*a*pinverse(a) = \n%f\n", pinv*a*pinv);
}
```

Output4

```
rank(a) = 2
pinverse(a) =
0.128000 -0.104000
0.030769 0.061538
-0.142154 0.235692

a*pinverse(a)*a =
7.000000 8.000000 1.000000
3.000000 6.000000 4.000000

pinverse(a)*a*pinverse(a) =
0.128000 -0.104000
0.030769 0.061538
-0.142154 0.235692
```

Example5
Calculate the Moore-Penrose pseudo inverse of a 3-by-2 matrix.

```
#include <numeric.h>
#define M 3
#define N 2
/* pseudoinverse */
int main() {
     array double a[M][N] = {7, 8,
                             1, 3,
                             6, 4}; /* m-by-n matrix */
    int r;
    array double pinv[N][M];

    r = rank(a);
    pinv = pinverse(a);
    printf("rank(a) = %d\n", r);
    printf("pinverse(a) = \n%f\n", pinv);
    printf("a*pinverse(a)*a = \n%f\n", a*pinv*a);
    printf("pinverse(a)*a*pinverse(a) = \n%f\n", pinv*a*pinv);
}
```

Output5

```
rank(a) = 2
pinverse(a) =
-0.053595 -0.209150 0.264052
0.139869 0.228758 -0.201307

a*pinverse(a)*a =
7.000000 8.000000
1.000000 3.000000
```

```
6.000000 4.000000

pinverse(a)*a*pinverse(a) =
-0.053595 -0.209150 0.264052
0.139869 0.228758 -0.201307
```

See Also

rank(), **svd**(), **qrdecomp**(), **inverse**().
cinverse().

References

polycoef

Synopsis
#include <**numeric.h**>
int polycoef(array double complex *p*[&], **array double complex** *x*[&]);

Purpose
Calculates the coefficients of the polynomial with specified roots.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
p An output array containing the calculated coefficients of the polynomial.

x An input one-dimensional array containing the roots of the polynomial.

Description
This function calculates the coefficients of a polynomial with specified roots expressed as an array *x*. The coefficients are in the order of descending powers. The polynomial it represents is $p_0x^n + ... + p_{n-1}x + p_n$.

Example1
Calculate the coefficients of polynomials

$$x^3 - 6x^2 - 72x - 27 = 0$$

and

$$x^4 - 12x^3 - 25x + 116 = 0$$

The roots of the polynomials are calculated by function **roots**().

```
#include <numeric.h>
int main() {
     array double x1[3], p1[4]  = {1, -6, -72, -27}; /* x^3-6x^2-72x-27 =0*/
     array double x2[4], p2[5]  = {1, -12, 0, 25, 116}; /* x^4-12x^3-25x+116 =0 */
     array double complex z2[4], zp2[5];
     array double complex z3[4], p3[5] = {complex(3,4), complex(4,2),
                            complex(5,3), complex(2,4), complex(1,5)};
     int status;


     roots(x1, p1);
     polycoef(p1, x1);
     printf("p1 from polycoef(p1, x1) = %0.2f\n", p1);
```

```
     /* x^4-12x^3-25x+116 =0  has two complex roots and two real roots */
     roots(x2, p2);
     roots(z2, p2);
     polycoef(p2, x2);
     printf("p2 from polycoef(p2, x2) =  %0.2f\n", p2);
     polycoef(zp2, z2);
     printf("zp2 from polycoef(zp2, z2) =\n %0.2f\n", zp2);

     roots(z3, p3);
     status = polycoef(p3, z3);
     if(status == 0) {
       printf("p3 from polycoef(p3, z3) = \n %0.2f\n", p3);
       printf("complex(3,4)*p3 from polycoef(p3, z3) = \n %0.2f\n", complex(3,4)*p3);
       roots(z3, p3);
       printf("z3 from roots(z3, p3) = \n %0.2f\n", z3);
     }
     else
       printf("polycoef(p3, z3) failed\n");
}
```

Output1

```
p1 from polycoef(p1, x1) = 1.00 -6.00 -72.00 -27.00

p2 from polycoef(p2, x2) =  1.00 NaN NaN NaN NaN

zp2 from polycoef(zp2, z2) =
 complex(1.00,0.00) complex(-12.00,0.00) complex(0.00,0.00) complex(25.00,0.00)
complex(116.00,0.00)

p3 from polycoef(p3, z3) =
 complex(1.00,0.00) complex(0.80,-0.40) complex(1.08,-0.44) complex(0.88,0.16)
complex(0.92,0.44)

complex(3,4)*p3 from polycoef(p3, z3) =
 complex(3.00,4.00) complex(4.00,2.00) complex(5.00,3.00) complex(2.00,4.00)
complex(1.00,5.00)

z3 from roots(z3, p3) =
 complex(0.23,1.28) complex(0.43,-0.73) complex(-0.75,-0.71) complex(-0.70,0.55)
```

Example2
The coefficients of polynomial with linearly spaced roots are calculated.

```
#include <numeric.h>
/*  Wilkinson's famous example */
#define N 10
int main() {
     array double x[N], p[N+1];
     linspace(x, 1, N);
     printf("x = %f\n", x);
     polycoef(p, x);
     printf("p = %f\n", p);
     roots(x, p);
     printf("x = %f\n", x);
}
```

Output2

```
x = 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
10.000000

p = 1.000000 -55.000000 1320.000000 -18150.000000 157773.000000 -902055.000000
3416930.000000 -8409500.000000 12753576.000000 -10628640.000000 3628800.000000


x = 9.000000 10.000000 8.000000 7.000000 6.000000 5.000000 4.000000 3.000000 2.000000
1.000000
```

Example3
The roots of a characteristic polynomial are calculated by function **eigensystem**() from a matrix, and then the coefficients are calculated again using function **polycoef**().

```
#include <numeric.h>
int main() {
     array double a[3][3] = {0.8, 0.2, 0.1,
                             0.1, 0.7, 0.3,
                             0.1, 0.1, 0.6};
     array double evalues[3],evectors[3][3];
     array double p[4];

     eigensystem(evalues, NULL, a);
     polycoef(p, evalues);
     printf("evalues = %f\n", evalues);
     printf("characteristic polynomial of matrix a = %f\n", p);
}
```

Output3

```
evalues = 1.000000 0.600000 0.500000

characteristic polynomial of matrix a = 1.000000 -2.100000 1.400000 -0.300000
```

See Also

roots(), **charpolycoef**().

References

polyder

Synopsis
#include <numeric.h>
int polyder(array double complex *y*[&], **array double complex** *x*[&]);

Syntax
polyder(*y*, *x*)

Purpose
Take derivative of a polynomial.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
y A vector of coefficients of the derivative of the original polynomial.

x A vector of coefficients of the original polynomial.

Description
The vector of coefficient of polynomial *x* can be of any supported arithmetic data type and size. Conversion of the data to double complex type is performed internally. If vector *x* is of real type with size n, then the vector *y* for the derivative is of real type with size $(n-1)$. If vector *x* is of complex type, then the vector *y* for the derivative is of complex type.

Algorithm
Given a polynomial $P(t)$ with coefficients of x_i, for $i = 0, 1, \cdots, N$. That is

$$P(t) = x_0 t^n + x_1 t^{n-1} + x_2 t^{n-2} + \cdots + x_{n-1} t + x_n$$

Then the first derivative of this polynomial is

$$\begin{aligned} P'(t) &= n * x_0 t^{n-1} + (n-1) * x_1 t^{n-2} + (n-2) * x_2 t^{n-3} + \cdots + x_{n-1} \\ &= y_0 t^{n-1} + y_1 t^{n-1} + y_2 t^{n-3} + \cdots + y_{n-1} \end{aligned}$$

Example
In this example:
polynomial: $(1-i)x^2 + 2x + (3+i) = 0$
derivative : $(2+i2)x + 2 = 0$

polynomial: $x^2 + 2x + 3 = 0$
derivative : $2x + 2 = 0$

```
#include <stdio.h>
#include <numeric.h>

#define N 3        /* data array size */

int main() {
   int i;
   array double complex x[N],y[N-1];
   array double x1[N],y1[N-1];
   for (i=0;i<N;i++) {
      x[i]=complex(1+i,i-1);
      x1[i]=i+1;
   }

   printf("Coef. of polynomial b\n");
   printf("x=%6.3f\n", x);

   polyder(y,x);            /* derivative of x polynomial */
   printf("Coef. of derivative of polynomial of b\n");
   printf("y=%6.3f\n", y);

   printf("Coef. of polynomial d\n");
   printf("x1=%6.3f",x1);

   polyder(y1,x1);            /* derivative of B polynomial */
   printf("Coef. of derivative of polynomial of d\n");

   printf("y1=%6.3f",y1);
}
```

Output

```
Coef. of polynomial b
x=complex( 1.000,-1.000) complex( 2.000, 0.000) complex( 3.000, 1.000)


Coef. of derivative of polynomial of b
y=complex( 2.000,-2.000) complex( 2.000, 0.000)

Coef. of polynomial d
x1= 1.000  2.000  3.000
Coef. of derivative of polynomial of d
y1= 2.000  2.000
```

See Also

polyder2().

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press, 1997.

polyder2()

Synopsis
#include <numeric.h>
int polyder2(array double complex *q*[&], **array double complex** *r*[&], **array double complex** *u*[&],
array double complex *v*[&]);

Syntax
polyder2(*q*, *NULL*, *u*, *v*)
polyder2(*q*, *r*, *u*, *v*)

Purpose
Calculate the derivative of product or quotient of two polynomials.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
q A vector contains the coefficients of the derivative of the product $(u * v)'$ of two polynomials *u* and *v* or coefficients of the numerator of the quotient $(u/v)'$ of two polynomials *u* and *v*.

r If input *r* is NULL the function implements the derivative of the product $(u * v)'$ of two polynomials *u* and *v*. Otherwise, it is a vector which contains the coefficients of the denominator from the derivative of the quotient $(u/v)'$ of two polynomials *u* and *v*.

u A vector of polynomial coefficients.

v A vector of polynomial coefficients.

Description
The operation of the derivative of a product or quotient of two polynomials *u* and *v* depends on argument *r*. If NULL is passed to argument *r*, functon **polyder**2() calculates the derivative of the product $(u * v)'$ of two polynomials *u* and *v*. Otherwise, it calculates the derivative of the quotient $(u/v)'$. The coefficient vector of polynomials *u* and *v* can be of any supported arithmetic data type with sizes n and m, respectively. Conversion of the data to double complex type is performed internally. If both vectors *u* and *v* are real type, the vector *q* with size $n + m - 2$ and *r* with size $2 * m - 1$ are real type. If one of vectors *u* and *v* is complex type, vectors *q* and *r* are complex type.

Algorithm
Given two polynomials $U(x)$ and $V(x)$

$$\begin{aligned} U(x) &= u_0 x^n + u_1 x^{n-1} + u_2 x^{n-2} + \cdots + u_{n-1} x + u_n \\ V(x) &= v_0 x^m + v_1 x^{m-1} + v_2 x^{m-2} + \cdots + v_{m-1} x + v_m \end{aligned}$$

The derivative of the product of two polynomials $U(x) * V(x)$ becomes

$$\begin{aligned} Q(x) &= (U(x)V(x))' = U'(x)V(x) + V'(x)U(x) \\ R(x) &= NULL \end{aligned}$$

The derivative of the quotient of two polynomials $U(x)/V(x)$ becomes

$$\left(\frac{U(x)}{V(x)}\right)' = \frac{Q(x)}{R(x)} = \frac{U'(x)V(x) - V'(x)U(x)}{V^2(x)}$$

where $Q(x)$ and $R(x)$ are the numerator and denominator of the derivative, respectively. The polynomials for $Q(x)$, applicable to derivative of either the product or quotient of two polynomials, and $R(x)$ can be represented as

$$\begin{aligned} Q(x) &= q_0 x^{n+m-1} + q_1 x^{n+m-2} + \cdots + q_{n+m-2}x + q_{n+m-1} \\ R(x) &= r_0 x^{2m-1} + r_1 x^{2m-2} + \cdots + r_{2m-2}x + r_{2m-1} \end{aligned}$$

Example
Given

$$\begin{aligned} U(x) &= x + 2 \\ V(x) &= x^2 + 2x \end{aligned}$$

then

$$\begin{aligned} (U(x)V(x))' &= 3x^2 + 8x + 4 \\ \left(\frac{U(x)}{V(x)}\right)' &= \frac{-x^2 - 4x - 4}{x^4 + 4x^3 + 4x^2} \end{aligned}$$

```
#include <stdio.h>
#include <numeric.h>

#define N 3        /* data array size */
#define M 2        /* response array size */

int main() {
   int i;
   array double complex v[N],u[M],q[N+M-2],r[2*N-1];
   array double v1[N],u1[M],q1[N+M-2],r1[2*N-1];
      v[0]=complex(1,1);v[1]=complex(2,2);v[2]=0;
      v1[0]=1;v1[1]=2;v1[2]=0;
   for (i=0;i<M;i++) {
      u[i]=complex(i+1,i);
      u1[i]=i+1;
   }

   printf("Coef. of polynomial v=%4.2f\n", v);
   printf("Coef. of polynomial u=%4.2f\n", u);

   polyder2(q, NULL, u, v);         /* derivative of v*u polynomial */
   printf("Coef. of deriv of poly v*u=%4.2f\n", q);
   printf("\n");
```

```
   polyder2(q, r, u, v);      /* derivative of u/v polynomial */
   printf("Coef. of derivative of polynomial of u/v\n");
   printf("Numerator: %4.2f\n", q);
   printf("Denormator:%4.2f\n", r);

   printf("Coef. of polynomial v=%6.3f\n", v1);
   printf("Coef. of polynomial u=%6.3f\n", u1);

   polyder2(q1, NULL, u1, v1);         /* derivative of b*a polynomial */
   printf("Coef. of derivative of polynomial of b*a=%6.3f\n", q1);
   printf("\n");

   polyder2(q1, r1, u1, v1);      /* derivative of a/b polynomial */
   printf("Coef. of derivative of polynomial of a/b=\n");
   printf("Numerator: %6.3f\n", q1);
   printf("Denormator:%6.3f\n", r1);
}
```

Output

```
Coef. of polynomial v=complex(1.00,1.00) complex(2.00,2.00) complex(0.00,0.00)

Coef. of polynomial u=complex(1.00,0.00) complex(2.00,1.00)

Coef. of deriv of poly v*u=complex(3.00,3.00) complex(6.00,10.00) complex(2.00,6.00)


Coef. of derivative of polynomial of u/v
Numerator: complex(-1.00,-1.00) complex(-2.00,-6.00) complex(-2.00,-6.00)

Denormator:complex(0.00,2.00) complex(0.00,8.00) complex(0.00,8.00) complex(0.00,0.00)
complex(0.00,-0.00)

Coef. of polynomial v= 1.000  2.000  0.000

Coef. of polynomial u= 1.000  2.000

Coef. of derivative of polynomial of b*a= 3.000  8.000  4.000


Coef. of derivative of polynomial of a/b=
Numerator: -1.000 -4.000 -4.000

Denormator: 1.000  4.000  4.000  0.000 -0.000
```

See Also

polyder().

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press, 1997.

polyeval

Synopsis
#include <numeric.h>
double polyeval(array double *c*[&], **double** *x*, ... /* [**array double** *dp*[&] */);

Syntax
polyeval(*c*, *x*)
polyeval(*c*, *x*, *dp*)

Purpose
Calculate the value of a polynomial and its derivatives.

Return Value
This function returns the value of a polynomial and its derivatives at a given point.

Parameters
c A vector of coefficients of a polynomial.

x A value of point x in which the value of polynomial is evaluated.

dp A vector of dimension m, it contains the values from the 1st to mth order derivatives of the polynomial at point *x*.

Description
The vector of coefficients of polynomial *c* can be of any supported real type and dimension n. The value of *x* can be of real type. Conversion of the data to double is performed internally. The return data is double type of the value of the polynomial. If the optional argument *dp* is specified, which shall be double type of dimension m, it passes back the values of 1st to mth order derivatives of the polynomial.

Algorithm
Given a polynomial

$$P(x) = c_0x^n + c_1x^{n-1} + \cdots + c_{n-3}x^3 + c_{n-2}x^2 + c_{n-1}x + c_n$$

The calculation of the polynomial value and its derivatives are implemented using the following algorithm

$$\begin{aligned} P &= (((\cdots(c_0x + c_1)x + \cdots + c_{n-3})x + c_{n-2})x + c_{n-1})x + c_n \\ P' &= ((\cdots(nc_0x + (n-1)c_1)x + \cdots + 3c_{n-3})x + 2c_{n-2})x + c_{n-1} \\ P'' &= (\cdots(n(n-1)c_0x + (n-1)(n-2)c_1)x + \cdots + 3 * 2c_{n-3})x + 2c_{n-2} \\ \vdots &= \vdots \quad \vdots \end{aligned}$$

$$\begin{aligned} P^{(n-1)} &= n(n-1)(n-2)\cdots 3*2c_0x + (n-1)(n-2)\cdots 3*2*1c_1 \\ P^{(n)} &= n(n-1)(n-2)\cdots 3*2*1c_0 \\ P^{(n+1)} &= 0 \end{aligned}$$

Example

Calculate the value of the polynormal

$$C(x) = x^5 - 5x^4 + 10x^3 - 10x^2 + 5x - 1$$

at point $x = 0.1$ and its first, second and third order derivatives.

```
#include <numeric.h>
#include <stdio.h>

#define NC 6
#define ND 3
#define NP 10

int main() {
   int i,j,k;
   float c[NC]={1.0,-5.0,10.0,-10.0,5.0,-1.0};
   float vp[ND], d[ND+1],x;
   string_t title = "   x    polynomial    first deriv   second deriv    third deriv";

   x=0.1;
   d[0] = polyeval(c,x,vp);
   for (j=0;j<ND;j++) d[j+1]=vp[j];
   printf("%s\n", title);
   printf("%4.3f",x);
   for (i=0;i<=ND;i++) {
      printf("%14.6f",d[i]);
   }
   printf("\n\n");

   x=0.1;
   printf("x=%f\tf(x)=%f\n",x,polyeval(c,x,vp));
}
```

Output

```
   x    polynomial    first deriv   second deriv    third deriv
0.100     -0.590490      3.280500    -14.580000     48.599998

x=0.100000 f(x)=-0.590490
```

see also

polyevalarray(), **cpolyeval**().

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press,1997.

polyevalarray

Synopsis
#include <**numeric.h**>
int polyevalarray(**array double complex** &*val*, *c*[&], &*x*);

Syntax
polyevalarray(*val*, *c*, *x*)

Purpose
Polynomial evaluation for a sequence of points.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
val The array of output which contains the polynomial evaluation values for points *x*. The dimension of array *val* is the same as that of **array** *x*.

c A vector of coefficients for the polynomial.

x An array which contains the value of points in which the polynomial is evaluated.

Description
The vector *c* with the coefficients of a polynomial can be of any supported arithmetic data type and size. The value of *x* can be any arithmetic type. Conversion of the data to double complex is performed internally. Array *val*, the same dimension and size as *x*, contains the evaluation results. If both *c* and *x* are real type, *val* is real type. Otherwise, it shall be complex type.

Algorithm
Given polynomial

$$Y(x_i) = c_0 x_i^n + c_1 x_i^{n-1} + c_2 x_i^{n-1} + \cdots + c_{n-1} x_i + c_n$$

the polynomial evaluation is implemented

$$Y_i = (((\cdots(c_0 x_i + c_1)x_i + c_2)x_i + c_3)x_i + \cdots + c_{n-1})x_i + c_n$$

where subscript i traverses all elements of array *x*.

Example
Evaluate polynormal

$$C(x) = x^5 - 5x^4 + 10x^3 - 10x^2 + 5x - 1$$

at points $x = \{0.1, 0.2, \cdots, 1.9, 2.0\}$.

```
#include <numeric.h>
#include <stdio.h>

#define NC 6
#define N 10

int main()
{
   int i,j,k;
   array double c[NC]={1.0,-5.0,10.0,-10.0,5.0,-1.0};
   array double x[N],val[N];

   for (i=0; i<N; i++)
      x[i]=0.2*(i+1);
   printf("x value    function evalulate\n");
   polyevalarray(val,c,x);
   for (i=0; i<N; i++)
      printf("%f    %f\n",x[i],val[i]);
}
```

Output

```
x value    function evalulate
0.200000    -0.327680
0.400000    -0.077760
0.600000    -0.010240
0.800000    -0.000320
1.000000    0.000000
1.200000    0.000320
1.400000    0.010240
1.600000    0.077760
1.800000    0.327680
2.000000    1.000000
```

see also

polyeval(), **cpolyeval**().

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press, 1997.

polyevalm

Synopsis
#include <**numeric.h**>
int polyevalm(**array double complex** *y[&][&]*, **array double complex** *c[&]*, **array double complex** *x[&][&]*);

Syntax
polyevalm(*y*, *c*, *x*)

Purpose
Evaluate a real or complex polynomial with matrix argument.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Input square matrix. It contains data to be evaluated.

c A vector of coefficients of a polynomial. It can be **real** or **complex** data type.

y Output square matrix which contains data of the evaluated polynomial with given matrix argument *x*.

Description
This function evaluates the polynomial with given matrix argument *x*. In this function, The input square matrix *x* can be of any supported arithmetic data type. The polynomial is specified by the coefficient vector *c*, which can be any supported arithmetic data type. The output matrix *y* could be **real** or **complex** data type as required.

Example
Evaluation of real/complex polynomials of real/complex matrices.

```
#include <numeric.h>
#include <stdio.h>

#define NC 6
#define ND 3
#define N 20

int main()
{
   int i,j,k;
   array double c[4]={1,2,3,4};
   array double complex c1[4]={complex(2,1),2,3,4};
   array double x[3][3]={1,2,3,
                         3,4,5,
                         6,7,8};
```

```
   array double y[3][3];
   array double complex x1[3][3]={1,2,complex(2,3),
                        3,4,5,
                        6,7,complex(5,8)};
   array double complex y1[3][3];
   polyevalm(y,c,x);
   printf("x=\n%f\n",x);
   printf("y=\n%f\n",y);
   polyevalm(y1,c1,x1);
   printf("x1=\n%5.2f\n",x1);
   printf("y1=\n%5.2f\n",y1);


   return 0;
}
```

Output

```
x=
1.000000 2.000000 3.000000
3.000000 4.000000 5.000000
6.000000 7.000000 8.000000

y=
397.000000 501.000000 609.000000
729.000000 931.000000 1125.000000
1233.000000 1566.000000 1903.000000

x1=
complex( 1.00, 0.00) complex( 2.00, 0.00) complex( 2.00, 3.00)
complex( 3.00, 0.00) complex( 4.00, 0.00) complex( 5.00, 0.00)
complex( 6.00, 0.00) complex( 7.00, 0.00) complex( 5.00, 8.00)

y1=
complex(-82.00,685.00) complex(-64.00,878.00) complex(-706.00,603.00)

complex(849.00,1110.00) complex(1137.00,1361.00) complex(-195.00,1912.00)

complex(-12.00,2034.00) complex(105.00,2594.00) complex(-1864.00,2023.00)
```

See Also

logm(), **cfunm**(), **polyeval**(), **polyevalarray**(), **cpolyeval**(), **sqrtm**().

References

G. H. Golub, C. F. Van Loan, Matrix Computations Third edition, The Johns Hopkins University Press, 1996

polyfit

Synopsis
#include <**numeric.h**>
int polyfit(**double** *a*[&], **double** *x*[&], **double** *y*[&], ... /* [**double** *sig*[ ], **int** *ia*[ ],
double *covar*[ ][ ], **double** **chisq*] */);

Syntax
polyfit(*a, x, y*)
polyfit(*a, x, y, sig*)
polyfit(*a, x, y, sig, ia*)
polyfit(*a, x, y, sig, ia, covar*)
polyfit(*a, x, y, sig, ia, covar, chisq*)

Purpose
Fit a set of data points *x, y* to a polynomial function.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
a An output array which contains coefficients of polynomial.

x An array which contains variable values of the data set.

y An array which contains function values of the data set.

sig An input array which contains individual standard (measurement error) deviations of data set. It can be set to 1 if unknown.

ia An input array. Those components of nonzero entries shall be fitted, and those components of zero entries shall be held at their input values.

covar The Covariance matrix of fitting results, describing the fit of data to the model. See algorithm for definition.

chisq Chi-square of fitting.

Description
Given a set of data points expressed in arrays *x* and *y* with the same dimension, use χ^2 minimization to fit the data set using a polynomial coefficients *a*. The fitted coefficients are passed by array *a*. The program also passes χ^2, covariance matrix *covar*. If the parameters of *ia* are held fixed, the corresponding components of covariance matrix will be zero.

Algorithm

The algorithm of the function **polyfit**() is based on that of the function **curvefit**(). For polynomial fitting, the base functions for **curvefit**() are set to the terms of polynomials internally. The general form of the fitting formula is

$$y(x) \quad = \quad \sum_{k=1}^{M} a_k x^k$$

The coefficients a_k are determined by minimizing chi-square which is defined as

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - \sum_{k=1}^{M} a_k x_i^k)}{\sigma_i} \right)^2$$

where σ_i is the standard deviation of the ith data point. If the standard deviations are unknown, they can be set to the constant value $\sigma = 1$. By defining a $n \times m$ matrix α with element (k, j) as

$$\alpha_{kj} = \sum_{i=1}^{N} \frac{x_i^j x_i^k}{\sigma_i^2}$$

The element (k, j) of the covariance matrix can be expressed as

$$cover_{kj} = [\alpha]_{kj}^{-1}$$

Example1

Fit the data set created by the polynomial

$$8x^4 + 5x^3 + 3x^2 + 6x + 7$$

The deviation with a fraction of a uniform random number is added to data set.

```
#include <numeric.h>
#include <stdio.h>
#define NPT 100                 /* Number of data points */
#define NTERM 5                 /* Number of terms */

int main() {
    int i,j,status;
    array double  u[NPT],x[NPT],y[NPT],a[NTERM];

    /* Create a data set of NTERM order polynomial with uniform random deviation*/
    linspace(x,0.1,0.1*NPT);
    y = 8*x.*x.*x.*x + 5*x.*x.*x + 3*x.*x + 6*x  + (array double[NPT])(7);

    /* put uniform random deviation on data set */
    urand(u);
    y += 0.1*u;
    status=polyfit(a,x,y);
    if(status) printf("Abnormal fit");
    printf("\n%11s\n","Coefficients");
    for (i=0;i<NTERM;i++)
        printf("  a[%1d] =  %8.6f \n",i,a[i]);
    printf("y = %f at x = %f\n", polyeval(a, 2.0), 2.0);
}
```

Output1

```
Coefficients
  a[0] =  8.000048
  a[1] =  4.999537
  a[2] =  2.998743
  a[3] =  6.015915
  a[4] =  7.033636
y = 199.057500 at x = 2.000000
```

Example2

Fit the data set created by the polynomial

$$8x^4 + 5x^3 + 3x^2 + 6x + 7$$

The deviation with a fraction of a uniform random number is added to data set. This example uses all the options of arguments of function **polyfit**(). The coefficients $a[1]$ and $a[3]$ are fixed and the standard deviation of data set is assumed to be a constant 0.1. The uncertainty which is defined as $dev[i] = \sqrt{covar[i][i]}$ is calculated.

```
#include <stdio.h>
#include <numeric.h>
#define NPT 100                 /* Number of data points */
#define NTERM 5                 /* Number sof term */
#define SPREAD 0.1              /* Number sof term */

int main() {
    int i,j,status,ia[NTERM];
    array double u[NPT],x[NPT],y[NPT],a[NTERM],dev[NTERM];
    array double sig[NPT],covar[NTERM][NTERM];
    double chisq;

    /* Create a data set of NTERM order polynomial with gaussian deviation*/
    linspace(x,0.1,0.1*NPT);
    y = 8*x.*x.*x.*x + 5*x.*x.*x + 3*x.*x + 6*x  + (array double[NPT])(7);

    /* put uniform random deviation on data set */
    urand(u);
    y += 0.1*u;
    sig=(array double[NPT])(SPREAD);

    ia[0] = 1; ia[1] = 0; ia[2] = 1; ia[3] = 0, ia[4] = 1;
    a[1] = 5; a[3] = 6;

    status=polyfit(a,x,y,sig,ia,covar,&chisq);
    dev = sqrt(diagonal(covar));
    if(status) printf("Abnormal fit");
    printf("\n%11s%23s\n","Coefficients");
    printf("\n%11s %21s\n","parameter","uncertainty");
    for (i=0;i<NTERM;i++)
        printf("  a[%1d] = %8.6f %12.6f\n",
            i,a[i],dev[i]);
    printf("chi-square = %12f\n",chisq);
    printf("full covariance matrix\n");
    printf("%12f",covar);
}
```

Output2

```
Coefficients
  parameter           uncertainty
  a[0] = 8.000007     0.000013
  a[1] = 5.000000     0.000000
  a[2] = 2.999413     0.001165
  a[3] = 6.000000     0.000000
  a[4] = 7.057378     0.018870
chi-square =     7.414554
full covariance matrix
    0.000000     0.000000    -0.000000     0.000000     0.000000
    0.000000     0.000000     0.000000     0.000000     0.000000
   -0.000000     0.000000     0.000001     0.000000    -0.000016
    0.000000     0.000000     0.000000     0.000000     0.000000
    0.000000     0.000000    -0.000016     0.000000     0.000356
```

See Also

curvefit(), **polyeval**(), **std**().

References

William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery, *Numeric Recipes in C*, Second Edition, Cambridge University Press, 1997.

product

Synopsis
#include <**numeric.h**>
double product(**array double** &*a*, ... /* [**array double** *v*[:]] */);

Syntax
product(a)
product(a, v)

Purpose
Calculate products of all elements and products of the elements of each rows of an array.

Return Value
This function returns the product of all elements.

Parameters
a The input array for which the products are calculated.

v The output array which contains the products of each row of a two-dimensional array.

Description
This function calculates the product of all elements in an array of any dimension. If the array is a two-dimensional matrix, the function can calculate the products of each row. The product of all elements of the array is returned by the function and the products of the elements of each row are passed out by the argument *v*. The input array can be of any dimension and any arithmetic data types.

Example1
Calculate the products of all elements of arrays with different data types and dimensions.

```
#include <numeric.h>
int main() {
    double a[3] = {1,2,3};
    double b[2][3] = {1,2,3,4,5,6};
    int b1[2][3] = {1,2,3,4,5,6};
    double c[2][3][5];
    c[0][0][0] = 10;
    double prod;

    prod = product(a);
    printf("prod(a) = %f\n", prod);
    prod = product(b);
    printf("product(b) = %f\n", prod);
    prod = product(b1);
    printf("product(b1) = %f\n", prod);
    prod = product(c);
    printf("product(c) = %f\n", prod);
}
```

Output1

```
prod(a) = 6.000000
product(b) = 720.000000
product(b1) = 720.000000
product(c) = 0.000000
```

Example2

Calculate the products of the elements of each row of the two-dimensional arrays with different data types and dimensions.

```
#include <numeric.h>
int main() {
    double a[2][3] = {1,2,3,
                      4,5,6};
    array double b[3][4] = {1,2,3,4,
                            5,6,7,8,
                            1,2,3,4};
    array double productv1[2], productv2[3];

    product(a, productv1);
    printf("product(a, productv1) = %f\n", productv1);
    product(b,productv2);
    printf("product(b,productv2) = %f\n", productv2);
}
```

Output2

```
product(a, productv1) = 6.000000 120.000000

product(b,productv2) = 24.000000 1680.000000 24.000000
```

See Also

cproduct(), **cumprod**(), **mean**(), **median**(), **sum**().

References

qrdecomp

Synopsis
#include <numeric.h>
int qrdecomp(array double complex *a*[&][&], **array double complex** *q*[&][&],
array double complex *r*[&][&]);

Purpose
Perform the orthogonal-triangular QR decomposition of a matrix.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
a Input matrix.

q Output unitary matrix.

r Output upper triangular matrix.

Description
This function performs a QR decomposition of a matrix *a* so that $a = qr$. The output *q* is a unitary matrix and r is an upper triangular matrix. Assume the size of input matrix *a* is $m \times n$. If $m \leq n$, there is only one type of output matrices *q* and *r*. The size of output matrix *q* is $m \times m$ and matrix *r* is $m \times n$. If $m > n$, there are two types of output matrices *q* and *r*. One is full size in which matrix *q* is $m \times m$ and *r* is $m \times n$. Another is an economy size in which matrix *q* is $m \times n$ and *r* is $n \times n$. This function checks the size of arguments and automatically selects the corresponding output type.

Example1
Perform QR decomposition of $4 \times 4, 2 \times 4$ and 4×2 matrices. The 4×2 matrices have two different types of output corresponding to the argument sizes passed in.

```
#include <numeric.h>
int main() {
     int m = m, n = 4;
     array double a[4][4] = {1, 5, -7, 4,
                             3, 2,  5, 2,
                             3, 2,  5, 2,
                             3, 9,  5, 2}; /* a[m][n] */
     array double q[n][n];
     array double r[n][n];

     qrdecomp(a, q, r);
     printf("q = \n%f\n", q);
     printf("r = \n%f\n", r);
     printf("q*r = \n%f\n", q*r);

     m = 2;
     n = 4;
```

```
    array double a1[2][4] = {1, 5, -7, 4,
                             3, 2,  5, 2}; /* a[m][n] */
    array double q1[m][m];    // m<n
    array double r1[m][n];

    qrdecomp(a1, q1, r1);
    printf("q1^T*q1 = \n%f\n", transpose(q1)*q1);
    printf("q1 = \n%f\n", q1);
    printf("r1 = \n%f\n", r1);
    printf("q1*r1 = \n%f\n", q1*r1);

    array double a2[4][2] = {1, 5,
                            -7, 4,
                             3, 2,
                             5, 2}; /* a[m][n] */
    m = 4;
    n = 2;
    array double q2[m][n], q21[m][m];     // m>=n
    array double r2[n][n], r21[m][n];
    int status;

    status = qrdecomp(a2, q21, r21);
    if(status == 0) {
        printf("q21^T*q21 = \n%f\n", transpose(q21)*q21);
        printf("q21 = \n%f\n", q21);
        printf("r21 = \n%f\n", r21);
        printf("q21*r21 = \n%f\n", q21*r21);
    }
    else
        printf("error: numerical error in qrdecomp()\n");

    status = qrdecomp(a2, q2, r2);
    if(status == 0) {
        printf("q2^T*q2 = \n%f\n", transpose(q2)*q2);
        printf("q2 = \n%f\n", q2);
        printf("r2 = \n%f\n", r2);
        printf("q2*r2 = \n%f\n", q2*r2);
    }
    else
        printf("error: numerical error in qrdecomp()\n");
}
```

Output1

```
q =
-0.188982 0.511914 0.837991 0.000000
-0.566947 -0.405266 0.119713 0.707107
-0.566947 -0.405266 0.119713 -0.707107
-0.566947 0.639893 -0.518756 -0.000000

r =
-5.291503 -8.315218 -7.181325 -4.157609
0.000000 6.697548 -4.436592 1.706382
0.000000 0.000000 -7.262591 2.793304
0.000000 0.000000 0.000000 0.000000

q*r =
1.000000 5.000000 -7.000000 4.000000
3.000000 2.000000 5.000000 2.000000
3.000000 2.000000 5.000000 2.000000
```

```
3.000000 9.000000 5.000000 2.000000

q1^T*q1 =
1.000000 0.000000
0.000000 1.000000

q1 =
-0.316228 -0.948683
-0.948683 0.316228

r1 =
-3.162278 -3.478505 -2.529822 -3.162278
0.000000 -4.110961 8.221922 -3.162278

q1*r1 =
1.000000 5.000000 -7.000000 4.000000
3.000000 2.000000 5.000000 2.000000

q21^T*q21 =
1.000000 0.000000 0.000000 0.000000
0.000000 1.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 1.000000

q21 =
-0.109109 -0.730552 -0.408074 -0.536530
0.763763 -0.491027 0.173257 0.381499
-0.327327 -0.323359 0.873744 -0.157694
-0.545545 -0.347312 -0.200072 0.736021

r21 =
-9.165151 0.763763
0.000000 -6.958209
0.000000 0.000000
0.000000 0.000000

q21*r21 =
1.000000 5.000000
-7.000000 4.000000
3.000000 2.000000
5.000000 2.000000

q2^T*q2 =
1.000000 0.000000
0.000000 1.000000

q2 =
-0.109109 -0.730552
0.763763 -0.491027
-0.327327 -0.323359
-0.545545 -0.347312

r2 =
-9.165151 0.763763
0.000000 -6.958209

q2*r2 =
1.000000 5.000000
-7.000000 4.000000
```

```
3.000000 2.000000
5.000000 2.000000
```

Example2
Perform QR decomposition of 2×4 complex matrix

```
#include <numeric.h>
int main() {
     array double complex a[2][4] = {1, 5, -7,  4,
                                     3, 3,  2, -13}; /* a[m][n] */
     array double complex a1[2][4] = {complex(1,2), 5, -7,  4,
                                     3, 3,  2, -13}; /* a[m][n] */
     int m = 2, n = 4;
     array double complex q[m][m];    // in the case of m<n
     array double complex r[m][n] ;

     qrdecomp(a, q, r);
     printf("q^H*q = \n%f\n", conj(transpose(q))*q);
     printf("q = \n%f\n", q);
     printf("r = \n%f\n", r);
     printf("q*r = \n%f\n", q*r);

     qrdecomp(a1, q, r);
     printf("q^H*q = \n%f\n", conj(transpose(q))*q);
     printf("q = \n%f\n", q);
     printf("r = \n%f\n", r);
     printf("q*r = \n%f\n", q*r);

     m = 4;
     n = 2;
     array double complex a2[4][2] = {complex(1,2), complex(3,5),
                                      complex(-7,2), complex(3,4),
                                      complex(3,0), complex(2,3),
                                      complex(2,0), complex(2,-13)};
                                          // a[m][n]
     array double complex q2[m][m];     // in the case of m>=n
     array double complex r2[m][n] ;
     qrdecomp(a2, q2, r2);
     printf("q2^H*q2 = \n%f\n", conj(transpose(q2))*q2);
     printf("q2 = \n%f\n", q2);
     printf("r2 = \n%f\n", r2);
     printf("q2*r2 = \n%f\n", q2*r2);
}
```

Output2

```
q^H*q =
complex(1.000000,0.000000) complex(0.000000,0.000000)
complex(0.000000,0.000000) complex(1.000000,0.000000)

q =
complex(-0.316228,0.000000) complex(-0.948683,0.000000)
complex(-0.948683,-0.000000) complex(0.316228,0.000000)

r =
complex(-3.162278,0.000000) complex(-4.427189,0.000000) complex(0.316228,0.000000)
complex(11.067972,0.000000)
complex(0.000000,0.000000) complex(-3.794733,0.000000) complex(7.273239,0.000000)
complex(-7.905694,0.000000)
```

```
q*r =
complex(1.000000,0.000000) complex(5.000000,0.000000) complex(-7.000000,0.000000)
complex(4.000000,0.000000)
complex(3.000000,0.000000) complex(3.000000,0.000000) complex(2.000000,0.000000)
complex(-13.000000,0.000000)

q^H*q =
complex(1.000000,0.000000) complex(-0.000000,0.000000)
complex(-0.000000,-0.000000) complex(1.000000,0.000000)

q =
complex(-0.267261,-0.534522) complex(0.717137,-0.358569)
complex(-0.801784,0.000000) complex(-0.000000,0.597614)

r =
complex(-3.741657,0.000000) complex(-3.741657,2.672612) complex(0.267261,-3.741657)
complex(9.354143,2.138090)
complex(0.000000,0.000000) complex(3.585686,0.000000) complex(-5.019960,-3.705209)
complex(2.868549,9.203260)

q*r =
complex(1.000000,2.000000) complex(5.000000,0.000000) complex(-7.000000,0.000000)
complex(4.000000,-0.000000)
complex(3.000000,-0.000000) complex(3.000000,0.000000) complex(2.000000,0.000000)
complex(-13.000000,-0.000000)

q2^H*q2 =
complex(1.000000,0.000000) complex(0.000000,0.000000) complex(0.000000,-0.000000)
complex(-0.000000,0.000000)
complex(0.000000,-0.000000) complex(1.000000,0.000000) complex(0.000000,-0.000000)
complex(0.000000,0.000000)
complex(0.000000,0.000000) complex(0.000000,0.000000) complex(1.000000,0.000000)
complex(-0.000000,0.000000)
complex(-0.000000,-0.000000) complex(0.000000,-0.000000) complex(-0.000000,-0.000000)
complex(1.000000,0.000000)

q2 =
complex(-0.118678,-0.237356) complex(-0.097267,-0.380224) complex(-0.535487,-0.061199)
complex(0.376291,-0.586055)
complex(0.830747,-0.237356) complex(-0.175866,0.098249) complex(0.279893,0.090757)
complex(0.027154,-0.354437)
complex(-0.356034,0.000000) complex(-0.110039,-0.362539) complex(0.784519,0.011697)
complex(0.238045,-0.239634)
complex(-0.237356,0.000000) complex(-0.119864,0.804660) complex(0.041031,0.075111)
complex(0.490320,-0.184604)

r2 =
complex(-8.426150,0.000000) complex(-1.186782,6.171265)
complex(0.000000,0.000000) complex(-14.335517,0.000000)
complex(0.000000,0.000000) complex(0.000000,0.000000)
complex(0.000000,0.000000) complex(0.000000,0.000000)

q2*r2 =
complex(1.000000,2.000000) complex(3.000000,5.000000)
complex(-7.000000,2.000000) complex(3.000000,4.000000)
complex(3.000000,-0.000000) complex(2.000000,3.000000)
complex(2.000000,0.000000) complex(2.000000,-13.000000)
```

See Also

orthonormalbase(), **ludecomp**(), **nullspace**() **qrdelete**(), **qrinsert**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

qrdelete

Synopsis
#include <**numeric.h**>
int qrdelete(**array double complex** *qout*[&][&], **array double complex** *rout*[&][&],
array double complex *q*[&][&], **array double complex** *r*[&][&], **int** *j*);

Purpose
Remove a column in QR factorized matrix.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
qout Output Q matrix in QR factorization of the matrix with a column removed.

rout Output R matrix in QR factorization of the matrix with a column removed.

q Input Q matrix in QR factorization of the original matrix.

r Input R matrix in QR factorization of the original matrix.

j Index specifying which column to be removed.

Description
This function changes Q and R to be the factorization of the matrix a with one of its columns deleted. Assume that Q and R is the original factorization of the matrix a which has n columns, this function returns $qout$ and $rout$ that are the factorization of the matrix obtained by removing the $(j+1)$th column of a.

Example1
Change QR decomposition of 6×6 complex matrix a to the decomposition of the matrix obtained by removing the third column of a.

```
#include <numeric.h>
#include <stdio.h>
int main() {
    array double complex a[6][6] = {1,  2,  3,   7.8, 8.9, 10,
                                    4,  5,  6,   1.2, 2.3, 3,
                                    7,  8,  9,   4.5, 5.6, 6,
                                    11, 15, 2.5, 5.5, 8.5, 0.1,
                                    13, 10, 7.5, 4.5, 1.5, 10,
                                    3.5,6.5,9.5, 10.5,0.2, 31};
    array double complex q[6][6], r[6][6];
    array double complex qout[6][6], rout[6][5];

    qrdecomp(a, q, r);
    qrdelete(qout, rout, q, r, 2);

    printf("qout = %.4f\nrout = %.4f\n", qout, rout);
```

```
    printf("qout*rout = %.4f\n", qout*rout);

    return 0;
}
```

Output1

```
qout = complex(-0.0521,0.0000) complex(-0.1565,0.0000) complex(0.6493,0.0000)
complex(0.7075,0.0000) complex(-0.0466,0.0000) complex(0.2201,0.0000)

complex(-0.2084,-0.0000) complex(-0.1186,0.0000) complex(-0.1817,0.0000)
complex(0.0216,0.0000) complex(0.8093,0.0000) complex(0.5040,0.0000)

complex(-0.3648,-0.0000) complex(-0.0807,0.0000) complex(0.0181,0.0000)
complex(0.2167,0.0000) complex(0.3968,0.0000) complex(-0.8098,0.0000)

complex(-0.5732,-0.0000) complex(-0.5377,0.0000) complex(-0.4142,0.0000)
complex(0.1481,0.0000) complex(-0.4111,0.0000) complex(0.1408,0.0000)

complex(-0.6774,-0.0000) complex(0.6717,0.0000) complex(0.1903,0.0000)
complex(-0.1300,0.0000) complex(-0.1219,0.0000) complex(0.1479,0.0000)

complex(-0.1824,-0.0000) complex(-0.4633,0.0000) complex(0.5807,0.0000)
complex(-0.6428,0.0000) complex(0.0396,0.0000) complex(-0.0112,0.0000)


rout = complex(-19.1898,0.0000) complex(-20.6229,0.0000) complex(-10.4144,0.0000)
complex(-8.9110,0.0000) complex(-15.8209,0.0000)
complex(0.0000,0.0000) complex(-5.9115,0.0000) complex(-6.5255,0.0000)
complex(-5.7735,0.0000) complex(-10.1036,0.0000)
complex(0.0000,0.0000) complex(0.0000,0.0000) complex(9.6041,0.0000)
complex(2.3431,0.0000) complex(25.9217,0.0000)
complex(0.0000,0.0000) complex(0.0000,0.0000) complex(0.0000,0.0000)
complex(8.4954,0.0000) complex(-12.7715,0.0000)
complex(0.0000,0.0000) complex(0.0000,0.0000) complex(0.0000,0.0000)
complex(0.0000,0.0000) complex(4.3108,0.0000)
complex(0.0000,0.0000) complex(0.0000,0.0000) complex(0.0000,0.0000)
complex(0.0000,0.0000) complex(0.0000,0.0000)

qout*rout = complex(1.0000,0.0000) complex(2.0000,0.0000) complex(7.8000,0.0000)
complex(8.9000,0.0000) complex(10.0000,0.0000)
complex(4.0000,0.0000) complex(5.0000,0.0000) complex(1.2000,0.0000)
complex(2.3000,0.0000) complex(3.0000,0.0000)
complex(7.0000,0.0000) complex(8.0000,0.0000) complex(4.5000,0.0000)
complex(5.6000,0.0000) complex(6.0000,0.0000)
complex(11.0000,0.0000) complex(15.0000,0.0000) complex(5.5000,0.0000)
complex(8.5000,0.0000) complex(0.1000,0.0000)
complex(13.0000,0.0000) complex(10.0000,0.0000) complex(4.5000,0.0000)
complex(1.5000,0.0000) complex(10.0000,0.0000)
complex(3.5000,0.0000) complex(6.5000,0.0000) complex(10.5000,0.0000)
complex(0.2000,0.0000) complex(31.0000,0.0000)
```

Example2
Change QR decomposition of 4×4 complex matrix a to the decomposition of the matrix obtained by removing the first column of a.

```
#include <numeric.h>
#include <stdio.h>
int main() {
```

```
    array double complex a[4][4] = {
       -0.7605, -0.2862, complex(-0.2748, 0.6488),  complex(-0.2748,-0.6488),
       -0.4394, 0.6886,  complex(-0.4194, -0.4555), complex(-0.4194,0.4555),
       0.1572, 0.3920,   complex(0.0360, 0.0915),   complex(0.0360,-0.0915),
       -0.4514, -0.5388, complex(-0.3322,-0.0089),  complex(-0.3322, 0.0089)
    };
    array double complex q[4][4], r[4][4];
    array double complex qout[4][4], rout[4][3];

    qrdecomp(a, q, r);
    qrdelete(qout, rout, q, r, 0);
    printf("qout = %.4f\nrout = %.4f\n", qout, rout);
    printf("qout*rout = %.4f\n", qout*rout);

    return 0;
}
```

Output2

```
qout = complex(-0.2862,0.0000) complex(-0.3148,0.5825) complex(-0.3580,-0.5464)
complex(0.2299,0.0000)
complex(0.6886,0.0000) complex(-0.4589,-0.1572) complex(-0.4933,0.1320)
complex(0.1727,-0.0000)
complex(0.3920,0.0000) complex(0.0480,0.3054) complex(0.0434,-0.2804)
complex(-0.8187,0.0000)
complex(-0.5388,0.0000) complex(-0.3843,-0.2882) complex(-0.4086,0.2550)
complex(-0.4970,0.0000)

rout = complex(1.0000,0.0000) complex(-0.0170,-0.4587) complex(-0.0170,0.4587)

complex(0.0000,0.0000) complex(0.8884,0.0000) complex(-0.0717,-0.0252)

complex(0.0000,0.0000) complex(0.0000,0.0000) complex(0.8852,0.0000)

complex(0.0000,0.0000) complex(0.0000,0.0000) complex(0.0000,0.0000)


qout*rout = complex(-0.2862,0.0000) complex(-0.2748,0.6488) complex(-0.2748,-0.6488)

complex(0.6886,0.0000) complex(-0.4194,-0.4555) complex(-0.4194,0.4555)

complex(0.3920,0.0000) complex(0.0360,0.0915) complex(0.0360,-0.0915)

complex(-0.5388,0.0000) complex(-0.3322,-0.0089) complex(-0.3322,0.0089)
```

Example3
The same as Example 1, except that all matrices contain real numbers.

```
#include <numeric.h>
#include <stdio.h>
int main() {
    array double a[6][6] = {1, 2, 3, 7.8, 8.9, 10,
                            4, 5, 6, 1.2, 2.3, 3,
                            7, 8, 9, 4.5, 5.6, 6,
                            11, 15, 2.5, 5.5, 8.5, .1,
                            13, 10, 7.5, 4.5, 1.5, 10,
                            3.5, 6.5, 9.5, 10.5, .2, 31};
    array double q[6][6], r[6][6];
```

```
    array double qout[6][6], rout[6][5];

    qrdecomp(a, q, r);
    qrdelete(qout, rout, q, r, 2);

    printf("qout = %.4f\nrout = %.4f\n", qout, rout);
    printf("qout*rout = %.4f\n", qout*rout);

    return 0;
}
```

Output3

```
qout = -0.0521 -0.1565 0.6493 0.7075 -0.0466 0.2201
-0.2084 -0.1186 -0.1817 0.0216 0.8093 0.5040
-0.3648 -0.0807 0.0181 0.2167 0.3968 -0.8098
-0.5732 -0.5377 -0.4142 0.1481 -0.4111 0.1408
-0.6774 0.6717 0.1903 -0.1300 -0.1219 0.1479
-0.1824 -0.4633 0.5807 -0.6428 0.0396 -0.0112

rout = -19.1898 -20.6229 -10.4144 -8.9110 -15.8209
0.0000 -5.9115 -6.5255 -5.7735 -10.1036
0.0000 0.0000 9.6041 2.3431 25.9217
0.0000 0.0000 0.0000 8.4954 -12.7715
0.0000 0.0000 0.0000 0.0000 4.3108
0.0000 0.0000 0.0000 0.0000 0.0000

qout*rout = 1.0000 2.0000 7.8000 8.9000 10.0000
4.0000 5.0000 1.2000 2.3000 3.0000
7.0000 8.0000 4.5000 5.6000 6.0000
11.0000 15.0000 5.5000 8.5000 0.1000
13.0000 10.0000 4.5000 1.5000 10.0000
3.5000 6.5000 10.5000 0.2000 31.0000
```

See Also

qrinsert(), **qrdecomp**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

qrinsert

Synopsis

#include <**numeric.h**>

int qrinsert(array double complex *qout*[&][&], **array double complex** *rout*[&][&],
array double complex *q*[&][&], **array double complex** *r*[&][&],
int *j*, **array double complex** *x*[&]);

Purpose

Insert a column in QR factorized matrix.

Return Value

This function returns 0 on success and -1 on failure.

Parameters

qout Output Q matrix in QR factorization of the matrix with an inserted column.

rout Output R matrix in QR factorization of the matrix with an inserted column.

q Input Q matrix in QR factorization of the original matrix.

r Input R matrix in QR factorization of the original matrix.

j Index specifying where to insert the extra column.

x The column to be inserted.

Description

This function changes Q and R to be the factorization of the matrix a with an extra column inserted. Assume that Q and R is the original factorization of the matrix a which has n columns, this function returns $qout$ and $rout$ that are the factorization of the matrix obtained by inserting an extra column x right before the $(j+1)$th column of a. If j is n, then x is inserted after the last column of a.

Example1

Change QR decomposition of 6×6 complex matrix a to the decomposition of the matrix obtained by inserting x before the third column of a.

```
#include <numeric.h>
#include <stdio.h>
int main() {
    array double complex a[6][6] = {1, 2, 3, 7.8, 8.9, 10,
                                    4, 5, 6, 1.2, 2.3, 3,
                                    7, 8, 9, 4.5, 5.6, 6,
                                    11, 15, 2.5, 5.5, 8.5, .1,
                                    13, 10, 7.5, 4.5, 1.5, 10,
                                    3.5, 6.5, 9.5, 10.5, .2, 31};
    array double complex q[6][6], r[6][6];
    array double complex qout[6][6], rout[6][7];
```

```
    array double complex x[6] = {10, 11, 12, 13, 14, 15};

    qrdecomp(a, q, r);
    qrinsert(qout, rout, q, r, 2, x);
    printf("qout = %.4f\nrout = %.4f\n", qout, rout);
    printf("qout*rout = %.4f\n", qout*rout);

    return 0;
}
```

Output1

```
qout = complex(-0.0521,0.0000) complex(-0.1565,0.0000) complex(0.5705,0.0000)
complex(-0.6514,0.0000) complex(-0.1766,0.0000) complex(0.4379,0.0000)

complex(-0.2084,-0.0000) complex(-0.1186,0.0000) complex(0.3449,0.0000)
complex(-0.0204,0.0000) complex(0.8873,0.0000) complex(-0.1891,0.0000)

complex(-0.3648,-0.0000) complex(-0.0807,0.0000) complex(0.1193,0.0000)
complex(0.6106,0.0000) complex(0.0177,0.0000) complex(0.6877,0.0000)

complex(-0.5732,-0.0000) complex(-0.5377,0.0000) complex(-0.5401,0.0000)
complex(-0.3008,0.0000) complex(-0.0048,0.0000) complex(-0.0062,0.0000)

complex(-0.6774,-0.0000) complex(0.6717,0.0000) complex(0.1117,0.0000)
complex(-0.1035,0.0000) complex(-0.1586,0.0000) complex(-0.2039,0.0000)

complex(-0.1824,-0.0000) complex(-0.4633,0.0000) complex(0.4869,0.0000)
complex(0.3180,0.0000) complex(-0.3950,0.0000) complex(-0.5078,0.0000)


rout = complex(-19.1898,0.0000) complex(-20.6229,0.0000) complex(-26.8632,0.0000)
complex(-12.9365,0.0000) complex(-10.4144,0.0000) complex(-8.9110,0.0000)
complex(-15.8209,0.0000)
complex(0.0000,0.0000) complex(-5.9115,0.0000) complex(-8.3741,0.0000)
complex(-2.6154,0.0000) complex(-6.5255,0.0000) complex(-5.7735,0.0000)
complex(-10.1036,0.0000)
complex(0.0000,0.0000) complex(0.0000,0.0000) complex(12.7767,0.0000)
complex(8.9680,0.0000) complex(8.0453,0.0000) complex(2.2131,0.0000)
complex(23.6127,0.0000)
complex(0.0000,0.0000) complex(0.0000,0.0000) complex(0.0000,0.0000)
complex(4.9123,0.0000) complex(-1.1385,0.0000) complex(-5.0738,0.0000)
complex(5.8830,0.0000)
complex(0.0000,0.0000) complex(0.0000,0.0000) complex(0.0000,0.0000)
complex(0.0000,0.0000) complex(-5.1202,0.0000) complex(0.2106,0.0000)
complex(-12.8281,0.0000)
complex(0.0000,0.0000) complex(0.0000,0.0000) complex(0.0000,0.0000)
complex(0.0000,0.0000) complex(0.0000,0.0000) complex(6.8539,0.0000)
complex(-9.8438,0.0000)

qout*rout = complex(1.0000,0.0000) complex(2.0000,0.0000) complex(10.0000,0.0000)
complex(3.0000,0.0000) complex(7.8000,0.0000) complex(8.9000,0.0000)
complex(10.0000,0.0000)
complex(4.0000,0.0000) complex(5.0000,0.0000) complex(11.0000,0.0000)
complex(6.0000,0.0000) complex(1.2000,0.0000) complex(2.3000,0.0000)
complex(3.0000,0.0000)
complex(7.0000,0.0000) complex(8.0000,0.0000) complex(12.0000,0.0000)
complex(9.0000,0.0000) complex(4.5000,0.0000) complex(5.6000,0.0000)
complex(6.0000,0.0000)
complex(11.0000,0.0000) complex(15.0000,0.0000) complex(13.0000,0.0000)
```

```
complex(2.5000,0.0000) complex(5.5000,0.0000) complex(8.5000,0.0000)
complex(0.1000,0.0000)
complex(13.0000,0.0000) complex(10.0000,0.0000) complex(14.0000,0.0000)
complex(7.5000,0.0000) complex(4.5000,0.0000) complex(1.5000,0.0000)
complex(10.0000,0.0000)
complex(3.5000,0.0000) complex(6.5000,0.0000) complex(15.0000,0.0000)
complex(9.5000,0.0000) complex(10.5000,0.0000) complex(0.2000,0.0000)
complex(31.0000,0.0000)
```

Example2
The same as Example1, Execpt that all matrices contain numbers.

```
#include <numeric.h>
#include <stdio.h>
int main() {
    array double a[6][6] = {1, 2, 3, 7.8, 8.9, 10,
                            4, 5, 6, 1.2, 2.3, 3,
                            7, 8, 9, 4.5, 5.6, 6,
                            11, 15, 2.5, 5.5, 8.5, .1,
                            13, 10, 7.5, 4.5, 1.5, 10,
                            3.5, 6.5, 9.5, 10.5, .2, 31};
    array double q[6][6], r[6][6];
    array double qout[6][6], rout[6][7];
    array double x[6] = {10, 11, 12, 13, 14, 15};

    qrdecomp(a, q, r);
    qrinsert(qout, rout, q, r, 2, x);
    printf("qout = %f\nrout = %f\n", qout, rout);
    printf("qout*rout = %.4f\n", qout*rout);

    return 0;
}
```

Output2

```
qout = -0.052111 -0.156528 0.570518 -0.651416 -0.176603 0.437905
-0.208444 -0.118631 0.344933 -0.020393 0.887315 -0.189091
-0.364776 -0.080733 0.119349 0.610631 0.017715 0.687745
-0.573220 -0.537685 -0.540132 -0.300849 -0.004805 -0.006178
-0.677442 0.671704 0.111661 -0.103480 -0.158569 -0.203877
-0.182388 -0.463268 0.486904 0.318045 -0.394971 -0.507826

rout = -19.189841 -20.622891 -26.863172 -12.936532 -10.414365 -8.910965 -15.820871

0.000000 -5.911545 -8.374077 -2.615442 -6.525484 -5.773472 -10.103602
0.000000 0.000000 12.776730 8.968006 8.045269 2.213064 23.612683
0.000000 0.000000 0.000000 4.912276 -1.138537 -5.073806 5.882958
0.000000 0.000000 0.000000 0.000000 -5.120200 0.210568 -12.828086
0.000000 0.000000 0.000000 0.000000 0.000000 6.853920 -9.843752

qout*rout = 1.0000 2.0000 10.0000 3.0000 7.8000 8.9000 10.0000
4.0000 5.0000 11.0000 6.0000 1.2000 2.3000 3.0000
7.0000 8.0000 12.0000 9.0000 4.5000 5.6000 6.0000
11.0000 15.0000 13.0000 2.5000 5.5000 8.5000 0.1000
13.0000 10.0000 14.0000 7.5000 4.5000 1.5000 10.0000
3.5000 6.5000 15.0000 9.5000 10.5000 0.2000 31.0000
```

See Also

qrdelete(), **qrdecomp**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

rank

Synopsis
#include <**numeric.h**>
int rank(**array double complex** *a*[&][&]);

Purpose
Find the rank of a matrix.

Return Value
This function returns the rank of a matrix.

Parameters
a Input two-dimensional array.

Description
This function provides an estimate of the number of linearly independent rows or columns of a matrix. The function can handle complex matrices.

Algorithm
The method used to compute the rank of a matrix in this function is based on the singular value decomposition by function **svd()**. The number of non-zero singular values with tolerance $tol = \max(m, n) \times \max(S) \times$ DBL_EPSILON.

Example1

```
#include <numeric.h>
int main() {
    array double a[2][2] = {2, 4,
                            3, 7};
    /* a2 is an ill-condition matrix  */
    array double a2[2][2] = {2, 4,
                             2.001, 4.0001};
    /* singular matrix */
    array float b[3][3] = {2, 4, 4,
                           3, 7, 3,
                           3, 7, 3};

    array double complex z[2][2] = {2, complex(4, 3),
                                    3, 7};
    /* a2 is an ill-condition matrix  */
    array double complex z2[2][2] = {2,     complex(4, 3),
                                     2.001, complex(4.0001,3)};
    array complex z3[2][2] = {2, complex(4, 3),
                              3, 7};
    int r;

    r = rank(a);
```

```
    printf("rank(a) = %d\n", r);

    r = rank(a2);
    printf("rank(a2) = %d\n", r);

    r = rank(b);
    printf("rank(b) = %d\n", r);

    r = rank(z);
    printf("rank(z) = %d\n", r);
    r = rank(z2);
    printf("rank(z2) = %d\n", r);
    r = rank(z3);
    printf("rank(z3) = %d\n", r);
}
```

Output1

```
rank(a) = 2
rank(a2) = 2
rank(b) = 2
rank(z) = 2
rank(z2) = 2
rank(z3) = 2
```

Example2

```
#include <numeric.h>
int main() {
    /* singular matrix */
    array float b[3][3] = {1, 2, 3,
                           4, 5, 6,
                           7, 8, 9};
    int r;

    r = rank(b);
    printf("rank(b) = %d\n", r);
    printf("determinant(b) = %f\n", determinant(b));
}
```

Output2

```
rank(b) = 2
determinant(b) = 0.000000
```

See Also

svd().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

rcondnum

Synopsis
#include <**numeric.h**>
double rcondnum(**array double complex** *a*[&][&]);

Purpose
Estimate reciprocal of the condition number of a matrix.

Return Value
This function returns the reciprocal of the condition number of a matrix.

Parameters
a Input matrix.

Description
This function calculates an estimate for the reciprocal of the condition of matrix *a* in 1-norm. If *a* is well conditioned , **rcondnum**(*a*) is near 1.0. If *a* is badly conditioned, **rcondnum**(*a*) is near 0.0. Compared to **condnum**(), function **rcondnum**() is a more efficient, but less reliable, method of estimating the condition of matrix.

Algorithm
The reciprocal of the condition number of a general real/complex matrix A is estimated in either the 1-norm or the infinity-norm, using the LU decomposition.

Example

```
#include <numeric.h>
int main() {
    array double a[2][2] = {2, 4,
                            3, 7};
    /* a2 is an ill-condition matrix  */
    array double a2[3][3] = {2, 4, 5,
                             2.001, 4.0001, 5.001,
                             6, 8, 3};
    array double complex z[2][2] = {2, 4,
                                    3, 7};
    array double complex z2[2][2] = {complex(2, 3), complex(4,2),
                                     3, 7};
    double rcond;

    rcond = rcondnum(a);
    printf("rcondnum(a) = %f\n", rcond);
    rcond = rcondnum(a2);
    printf("rcondnum(a2) = %f\n", rcond);

    rcond = rcondnum(z);
    printf("rcondnum(z) = %f\n", rcond);
    rcond = rcondnum(z2);
```

```
    printf("rcondnum(z2) = %f\n", rcond);
}
```

Output

```
rcondnum(a) = 0.018182
rcondnum(a2) = 0.000035
rcondnum(z) = 0.018182
rcondnum(z2) = 0.131909
```

See Also

condnum(), **norm**(), **rank**(), **svd**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

residue

Synopsis
#include <numeric.h>
int residue(**array double** *u*[&], **array double** *v*[&], **array double complex** *r*[&],
array double complex *p*[&], **array double** *k*[&]);

Syntax
residue(*u*, *v*, *r*, *p*, *k*)

Purpose
Partial-fraction expansion or residue computation.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
u A vector of size m which contains the numerator of the ratio of two polynomials.

v A vector of size n which contains the denominator of the ratio of two polynomials.

r A vector of size $(n-1)$ which contains the residues of a partial fraction expansion of the ratio of two polynomials.

p A vector of size $(n-1)$ which contains the poles of a partial fraction expansion of the ratio of two polynomials.

k A vector of size $(m-n+1)$ which contains the direct term of a partial fraction expansion of the ratio of two polynomials. If $m \leq n$, then *k* contains NULL.

Description
Function **residue**() finds the residues, poles and direct term of a partial fraction expansion of the ratio of two polynomials $V(s)$ and $U(s)$. Given the ratio of two polynomials

$$\frac{U(s)}{V(s)} = \frac{u_0 s^m + u_1 s^{m-1} + \cdots + u_{m-1}s + u_m}{s^n + v_1 s^{n-1} + \cdots + v_{n-1}s + v_n}$$

If there are no multiple roots, the expansion becomes

$$\frac{U(s)}{V(s)} = \frac{r_0}{s-p_0} + \frac{r_1}{s-p_1} + \cdots + \frac{r_{n-1}}{s-p_{n-1}} + K(s)$$

If p_i is a pole of multiplicity l, then the expansion becomes

$$\frac{U(s)}{V(s)} = \frac{r_0}{s-p_0} + \frac{r_1}{s-p_1} + \cdots + \frac{r_i}{(s-p_i)^l} + \cdots + \frac{r_{i+1}}{(s-p_i)^2} + \frac{r_{i+l}}{s-p_i} + \cdots + \frac{r_{n-1}}{s-p_{n-1}} + K(s)$$

where $K(x)$ is the direct term. If $m > n$, $K(s)$ is

$$K(s) = k_0 s^{m-n} + k_1 s^{m-n-1} + \cdots + k_{m-n} s + k_{m-n+1}$$

Otherwise, $K(s)$ is empty. Vectors *u* and *v* specify the coefficients of the polynomials in descending powers of s. They can be any real type. Conversion of the data to double type is performed internally. The residues *r* and poles *p* are any compatible data type according to the residue computation. If the argument of real type is passed and the result is complex type, the value of NaN will be passed out. The direct term *k* is always real type.

Algorithm

Given the ratio of two polynomials $U(x)$ and $V(x)$

$$\frac{U(s)}{V(s)} = \frac{u_0 s^m + u_1 s^{m-1} + \cdots + u_{m-1} s + u_m}{s^n + v_1 s^{n-1} + \cdots + v_{n-1} s + v_n}$$

If there are no multiple roots, it can be written as

$$\frac{U(s)}{V(s)} = \frac{U(s)}{(s-p_0)(s-p_1)\cdots(s-p_{n-1})}$$

where p_i can be real or complex number. Then it becomes

$$\frac{U(s)}{V(s)} = \frac{r_0}{s-p_0} + \frac{r_1}{s-p_1} + \cdots + \frac{r_{n-1}}{s-p_{n-1}}$$

where

$$r_i = [(s-p_i)\frac{U(s)}{V(s)}]_{s=p_i}$$

If l of the (n - 1) poles are identical, or say, the root at $s = p_i$ is of multiplicity l, then

$$\frac{U(s)}{V(s)} = \frac{U(s)}{(s-p_0)(s-p_1)\cdots(s-p_i)^l\cdots(s-p_i)^2(s-p_i)\cdots(s-p_{n-l})}$$

The equation can be expanded as

$$\frac{U(s)}{V(s)} = \frac{r_0}{s-p_0} + \frac{r_1}{s-p_1} + \cdots + \frac{r_i}{(s-p_i)^l} + \cdots + \frac{r_{i+1}}{(s-p_i)^2} + \frac{r_{i+l}}{s-p_i} + \cdots + \frac{r_{n-1}}{s-p_{n-l}}$$

where the coefficients correspond to simple poles could be calculated by the method described above. The determination of the coefficients that correspond to the multiple-order is given below.

$$r_{i+k} = \frac{1}{(k-1)!}\frac{d^{k-1}}{ds^{k-1}}[(s-p_i)^l\frac{U(s)}{V(s)}]_{s=p_i} \qquad k = 1, 2, \cdots, l$$

Example 1

Partial-fraction expansion with single real roots and without direct term.

$$\frac{10s+6}{2s^3+12s^2+22s+12} = \frac{-1}{s+1} + \frac{7}{s+2} + \frac{-6}{s+3}$$

```
#include <stdio.h>
#include <numeric.h>

#define M 2        /* data array size */
#define N 4        /* response array size */

int main() {
   array double u[M] = {10, 6};
   array double v[N] = {2, 12, 22, 12};
   array double r[N-1],p[N-1];
   array double k[1];

   residue(u,v,r,p,k);
   printf("coef. of u=%f\n",u);
   printf("coef. of v=%f\n",v);
   printf("coef. of r=%f\n",r);
   printf("coef. of p=%f\n",p);
   printf("coef. of k=%f\n",k);
}
```

Output

```
coef. of u=10.000000 6.000000

coef. of v=2.000000 12.000000 22.000000 12.000000

coef. of r=-1.000000 7.000000 -6.000000

coef. of p=-1.000000 -2.000000 -3.000000

coef. of k=0.000000
```

Example 2

A partial-fraction expansion with single roots of complex numbers and without direct term.

$$\frac{x+3}{x^2+2x+5} = \frac{0.5+i0.5}{s+(1+i2)} + \frac{0.5-i0.5}{s+(1-i2)}$$

```
#include <stdio.h>
#include <numeric.h>

#define M 2        /* data array size */
#define N 3        /* response array size */

int main() {
   array double u[M] = {1, 3};
   array double v[N] = {1, 2, 5};
   array double complex r[N-1],p[N-1];
   array double r1[N-1],p1[N-1];
   array double k[1];

   residue(u,v,r,p,k);
   printf("coef. of u=%f\n",u);
   printf("coef. of v=%f\n",v);
   printf("coef. of r=%f\n",r);
   printf("coef. of p=%f\n",p);
   printf("coef. of k=%f\n",k);
```

```
   printf("\n r1, p1 pass by real data, but the result are complex data.\n");
   residue(u,v,r1,p1,k);
   printf("coef. of r=%f\n",r1);
   printf("coef. of p=%f\n",p1);
}
```

Output

```
coef. of u=1.000000 3.000000

coef. of v=1.000000 2.000000 5.000000

coef. of r=complex(0.500000,0.500000) complex(0.500000,-0.500000)

coef. of p=complex(-1.000000,-2.000000) complex(-1.000000,2.000000)

coef. of k=0.000000


 r1, p1 pass by real data, but the result are complex data.
coef. of r=NaN NaN

coef. of p=NaN NaN
```

Example 3

A partial-fraction expansion with single roots and has a direct term.

$$\frac{2s^3 + 12s^2 + 22s + 12}{10s + 6} = \frac{0.2688}{s + 0.6} + 0.2s^2 + 1.08s + 1.552$$

```
#include <stdio.h>
#include <numeric.h>

#define N 2        /* data array size */
#define M 4        /* response array size */

int main() {
   array double v[N] = {10, 6};
   array double u[M] = {2, 12, 22, 12};
   array double r[N-1],p[N-1];
   array double k[M-N+1];

   residue(u,v,r,p,k);
   printf("coef. of u=%f\n",u);
   printf("coef. of v=%f\n",v);
   printf("coef. of r=%f\n",r);
   printf("coef. of p=%f\n",p);
   printf("coef. of k=%f\n",k);
}
```

Output

```
coef. of u=2.000000 12.000000 22.000000 12.000000

coef. of v=10.000000 6.000000
```

```
coef. of r=0.268800

coef. of p=-0.600000

coef. of k=0.200000 1.080000 1.552000
```

Example 4

A partial-fraction expansion with multiplicity roots and without direct term.

$$\frac{s^2+2s+3}{s^5+5s^4+9s^3+7s^2+2s} = \frac{3}{2(s+2)} - \frac{3}{(s+1)^3} - \frac{2}{s+1} + \frac{3}{2s}$$

```
#include <stdio.h>
#include <numeric.h>

#define M 3        /* data array size */
#define N 6        /* response array size */

int main() {
   array double u[M] = {1,2,3};
   array double v[N] = {1, 5, 9, 7, 2, 0};
   array double r[N-1],p[N-1];
   array double k[1];

   residue(u,v,r,p,k);
   printf("coef. of u=%f\n",u);
   printf("coef. of v=%f\n",v);
   printf("coef. of r=%f\n",r);
   printf("coef. of p=%f\n",p);
   printf("coef. of k=%f\n",k);
}
```

Output

```
coef. of u=1.000000 2.000000 3.000000

coef. of v=1.000000 5.000000 9.000000 7.000000 2.000000 0.000000

coef. of r=1.500000 -3.000000 0.000000 -2.000000 1.500000

coef. of p=-2.000000 -1.000000 -1.000000 -1.000000 0.000000

coef. of k=0.000000
```

References

Benjamin C. Kuo, *Automatic Control System*, fifth edition, Prentice-Hall, 1987, p. 28-32.

roots

Synopsis
#include <**numeric.h**>
int roots(**array double complex** *x*[&], **array double complex** *p*[&]);

Purpose
Find the roots of a polynomial.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x An output which contains the roots of the polynomial.

p An input which contains the coefficients of the polynomial.

Description
This function finds the roots of the polynomial. The function can handle the polynomial with complex coefficients.

Algorithm
The roots of a polynomial are obtained by computing the eigenvalues of the corresponding companion matrix of the polynomial.

Example

```
#include <numeric.h>
int main() {
     array double x1[3], p1[4]  = {1, -6, -72, -27}; /* x^3-6x^2-72x-27 =0*/
     array double x2[4], p2[5]  = {1, -12, 0, 25, 116}; /* x^4-12x^3-25x+116 =0 */
     array double complex z2[4];
     array double complex z3[4], p3[5] = {complex(3,4), complex(4,2),
                          complex(5,3), complex(2,4), complex(1,5)};
     int status, i;


     roots(x1, p1);
     printf("x1 from roots(x1, p1) = %.3f\n", x1);
     for(i=0; i<3; i++)
       printf("polyeval(p1, x1[%d]) = %.3f\n", i,  polyeval(p1, x1[i]));


     /* x^4-12x^3-25x+116 =0  has two complex roots and two real roots */
     roots(x2, p2);
     printf("x2 from roots(x2, p2) = %.3f\n", x2);
     roots(z2, p2);
     printf("z2 from roots(z2, p2) = \n%.3f\n", z2);
```

```
    status = roots(z3, p3);
    if(status == 0)
      printf("z3 from roots(z3, p3) = \n%.3f\n", z3);
    else
      printf("roots(z3, p3) failed\n");

    for(i=0; i<4; i++)
      printf("cpolyeval(p3, z3[%d]) = %.3f\n", i,  cpolyeval(p3, z3[i]));
}
```

Output

```
x1 from roots(x1, p1) = 12.123 -5.735 -0.388

polyeval(p1, x1[0]) = 0.000
polyeval(p1, x1[1]) = -0.000
polyeval(p1, x1[2]) = 0.000
x2 from roots(x2, p2) = 11.747 2.703 NaN NaN

z2 from roots(z2, p2) =
complex(11.747,0.000) complex(2.703,0.000) complex(-1.225,1.467) complex(-1.225,-1.467)


z3 from roots(z3, p3) =
complex(0.226,1.281) complex(0.431,-0.728) complex(-0.754,-0.708) complex(-0.703,0.554)


cpolyeval(p3, z3[0]) = complex(0.000,-0.000)
cpolyeval(p3, z3[1]) = complex(0.000,-0.000)
cpolyeval(p3, z3[2]) = complex(0.000,-0.000)
cpolyeval(p3, z3[3]) = complex(0.000,-0.000)
```

See Also

fzero(), **residue**().

References

rot90()

Synopsis
#include <numeric.h>
int rot90(array double complex *y[&][&]*, **array double complex** *x[&][&]*, ... /* [**int** *k*] */);

Syntax
rot90(*y*, *x*)

Purpose
Rotate matrix 90 degrees.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Two-dimensional matrix which contains the original data.

y Two-dimensional matrix of the same data type and size correspondence with the size of rotation of matrix *x*. It contains the rotation of input matrix *x*.

k Integral data which indicates the $k*90$ degrees rotation of *x*. For a positive *k*, the matrix is rotated counter clockwise. For a negative *k*, the matrix is rotated clockwise.

Description
This function rotates matrix *x* $k*90$ degrees in counter clockwise direction.

Example

```
#include <numeric.h>
int main() {
   array double x[4][2]={1,3,
                         2.45,8.56,
                         3,5,
                         6,8};
   array double complex zx[3][2]={complex(1,1),0,
                         3,complex(4,1),
                         0,0};
   array double complex zy[3][2];
   array double y[2][4], y1[4][2];

   rot90(y,x);
   printf("x = \n%f",x);
   printf("y = rotate 90 degrees\n%f",y);
   rot90(y1,x,2);
   printf("y1 = rotate 180 degrees\n%f",y1);
   rot90(y1,x,8);
   printf("y1 = rotate 720 degrees\n%f",y1);
```

```
   rot90(y,x,-1);
   printf("y = totate -90/270 degrees\n%f",y);
   printf("\n");

   rot90(zy,zx,2);         /* complex data rotation */
   printf("zx = \n%5.1f",zx);
   printf("zy = \n%5.1f",zy);

}
```

Output

```
x =
1.000000 3.000000
2.450000 8.560000
3.000000 5.000000
6.000000 8.000000
y = rotate 90 degrees
3.000000 8.560000 5.000000 8.000000
1.000000 2.450000 3.000000 6.000000
y1 = rotate 180 degrees
8.000000 6.000000
5.000000 3.000000
8.560000 2.450000
3.000000 1.000000
y1 = rotate 720 degrees
1.000000 3.000000
2.450000 8.560000
3.000000 5.000000
6.000000 8.000000
y = totate -90/270 degrees
6.000000 3.000000 2.450000 1.000000
8.000000 5.000000 8.560000 3.000000

zx =
complex(  1.0,  1.0) complex(  0.0,  0.0)
complex(  3.0,  0.0) complex(  4.0,  1.0)
complex(  0.0,  0.0) complex(  0.0,  0.0)
zy =
complex(  0.0,  0.0) complex(  0.0,  0.0)
complex(  4.0,  1.0) complex(  3.0,  0.0)
complex(  0.0,  0.0) complex(  1.0,  1.0)
```

See Also

flipud(), **fliplr**().

rsf2csf

Synopsis
#include <**numeric.h**>
int rsf2csf(array double complex *uc*[:][:], **array double complex** *tc*[:][:],
array double complex *ur*[&][&], **array double complex** *tr*[&][&]);

Purpose
Convert a real upper quasi-triangular Schur form to a complex upper triangular Schur form.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
uc Output complex unitary matrix.

tc Output complex quasi-triangular Schur matrix.

ur Input real unitary matrix.

tr Input real upper triangular Schur matrix.

Description
This function converts a real upper quasi-triangular Schur form, which is the result of Schur decomposition, to a complex upper triangular Schur form. Assume that (u_r, t_r) is the real Schur form of the real square matrix x. This function can produce the complex Schur form (u_c, t_c) from the real one (u_r, t_r), such that $x = u_c t_c u_c'$. The matrix t_c is upper triangular with the eigenvalues of x on the diagonal.

Example
Transform the output of function **schurdecomp**() from real Schur form (ur, tr) to complex Schur form (uc, tc).

```
#include <array.h>

int main() {
    array double x1[4][4] = {1,1,1,3, 1,2,1,1, 1,1,3,1, -2,1,1,4};
    array double ur1[4][4], tr1[4][4];
    array double complex uc1[4][4], tc1[4][4];

    array double x2[3][3] = {2, 3, 4, 5, 6, 7, 8, 9, 10};
    array double ur2[3][3], tr2[3][3];
    array double complex uc2[3][3], tc2[3][3];

    schurdecomp(x1, ur1, tr1);
    printf("ur1 = %.4f\ntr1 = %.4f\n", ur1, tr1);

    rsf2csf(uc1, tc1, ur1, tr1);
    printf("uc1= %.4f\ntc1 = %.4f\n", uc1, tc1);
    printf("norm(uc1) = %.4f\n\n", norm(uc1, "2"));
```

```
    printf("uc1*tc1*uc1' = %.4f\n\n", uc1*tc1*transpose(conj(uc1)));

    schurdecomp(x2, ur2, tr2);
    printf("ur2 = %.4f\ntr2 = %.4f\n", ur2, tr2);

    rsf2csf(uc2, tc2, ur2, tr2);
    printf("uc2 = %.4f\ntc2 = %.4f\n", uc2, tc2);
    printf("norm(uc2) = %.4f\n\n", norm(uc2, "2"));
    printf("uc2*tc2*uc2' = %.4f\n\n", uc2*tc2*transpose(conj(uc2)));

    return 0;
}
```

Output

```
ur1 = 0.8835 0.3186 -0.0197 -0.3428
0.4458 -0.3651 0.1666 0.8001
0.0480 -0.5469 0.7191 -0.4260
-0.1355 0.6827 0.6743 0.2466

tr1 = 1.9202 0.9542 2.9655 0.8572
-2.2777 1.9202 1.1416 1.5630
0.0000 0.0000 4.8121 1.1314
0.0000 0.0000 0.0000 1.3474

uc1= complex(-0.2675,0.4801) complex(0.7417,-0.1731) complex(-0.0197,0.0000)
complex(-0.3428,0.0000)
complex(0.3065,0.2422) complex(0.3743,0.1984) complex(0.1666,0.0000)
complex(0.8001,0.0000)
complex(0.4591,0.0261) complex(0.0403,0.2971) complex(0.7191,0.0000)
complex(-0.4260,0.0000)
complex(-0.5732,-0.0736) complex(-0.1138,-0.3710) complex(0.6743,0.0000)
complex(0.2466,0.0000)

tc1 = complex(1.9202,1.4742) complex(1.3236,0.0000) complex(-0.9583,-1.6113)
complex(-1.3122,-0.4658)
complex(0.0000,0.0000) complex(1.9202,-1.4742) complex(2.4895,0.6203)
complex(0.7196,0.8493)
complex(0.0000,0.0000) complex(0.0000,0.0000) complex(4.8121,0.0000)
complex(1.1314,0.0000)
complex(0.0000,0.0000) complex(0.0000,0.0000) complex(0.0000,0.0000)
complex(1.3474,0.0000)

norm(uc1) = 1.0000

uc1*tc1*uc1' = complex(1.0000,-0.0000) complex(1.0000,-0.0000) complex(1.0000,-0.0000)
complex(3.0000,0.0000)
complex(1.0000,0.0000) complex(2.0000,-0.0000) complex(1.0000,0.0000)
complex(1.0000,0.0000)
complex(1.0000,0.0000) complex(1.0000,-0.0000) complex(3.0000,-0.0000)
complex(1.0000,-0.0000)
complex(-2.0000,-0.0000) complex(1.0000,0.0000) complex(1.0000,-0.0000)
complex(4.0000,0.0000)


ur2 = 0.2826 0.8680 0.4082
0.5383 0.2087 -0.8165
0.7939 -0.4505 0.4082

tr2 = 18.9499 4.8990 0.0000
```

```
0.0000 -0.9499 -0.0000
0.0000 0.0000 -0.0000

uc2 = complex(0.2826,0.0000) complex(0.8680,0.0000) complex(0.4082,0.0000)

complex(0.5383,0.0000) complex(0.2087,0.0000) complex(-0.8165,0.0000)

complex(0.7939,0.0000) complex(-0.4505,0.0000) complex(0.4082,0.0000)


tc2 = complex(18.9499,0.0000) complex(4.8990,0.0000) complex(0.0000,0.0000)

complex(0.0000,0.0000) complex(-0.9499,0.0000) complex(-0.0000,0.0000)

complex(0.0000,0.0000) complex(0.0000,0.0000) complex(-0.0000,0.0000)


norm(uc2) = 1.0000

uc2*tc2*uc2' = complex(2.0000,0.0000) complex(3.0000,0.0000) complex(4.0000,0.0000)

complex(5.0000,0.0000) complex(6.0000,0.0000) complex(7.0000,0.0000)

complex(8.0000,0.0000) complex(9.0000,0.0000) complex(10.0000,0.0000)
```

See Also

schurdecomp().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

schurdecomp

Synopsis
#include <**numeric.h**>
int schurdecomp(**array double complex** *a*[&][&], **array double complex** *q*[&][&],
array double complex *t*[&][&]);

Purpose
Compute Schur decomposition of a matrix.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
a An input two-dimensional array.

q An output two-dimensional array which contains *q* matrix.

t An output two-dimensional array which contains *t* matrix.

Description
This function computes Schur form of a square matrix. The Schur matrix **T** for a square matrix **A** of real type is defined as

$$A = QTQ^T$$

where matrix **Q** is an orthogonal matrix with Q^TQ=**I**. For matrix **A** of complex type, the Schur matrix *T* is defined as

$$A = QTQ^H$$

where matrix **Q** is unitary with Q^HQ=**I**.

Example1
Compute Schur decomposition of a real matrix.

```
#include <numeric.h>
int main() {
     int n = 2;
     array double a[2][2] = {8, -3,
                             -5, 9};
     array double t[n][n];
     array double q[n][n];
     int status;

     status = schurdecomp(a, q, t);
     if(status == 0) {
         printf("Schur vectors q =\n%f\n", q);
         printf("Schur form t =\n%f\n", t);
         printf("q*t*q^T =\n%f\n", q*t*transpose(q));
         printf("q^T*q =\n%f\n", transpose(q)*q);
```

```
    }
    else
        printf("error: numerical error in schurdecomp()\n");
    schurdecomp(a, q, NULL);
    printf("Schur vectors q =\n%f\n", q);
}
```

Output1

```
Schur vectors q =
0.661061 -0.750332
0.750332 0.661061

Schur form t =
4.594875 2.000000
0.000000 12.405125

q*t*q^T =
8.000000 -3.000000
-5.000000 9.000000

q^T*q =
1.000000 0.000000
0.000000 1.000000

Schur vectors q =
0.661061 -0.750332
0.750332 0.661061
```

Example2
Compute Schur decomposition of complex matrix.

```
#include <numeric.h>
int main() {
    int n = 2;
    array double complex a[2][2] = {8, -3,
                                    -5, 9};
    array double complex z[2][2] = {complex(1,2), -3,
                                    -5, 9};
    array double complex t[n][n];
    array double complex q[n][n];

    schurdecomp(a, q, t);
    printf("Schur vectors q =\n%f\n", q);
    printf("Schur form t =\n%f\n", t);
    printf("q*t*q^H =\n%f\n", q*t*conj(transpose(q)));
    printf("q^H*q =\n%f\n", conj(transpose(q))*q);


    schurdecomp(z, q, t);
    printf("Schur vectors q =\n%f\n", q);
    printf("Schur form t =\n%f\n", t);
    printf("q*t*q^H =\n%f\n", q*t*conj(transpose(q)));
    printf("q^H*q =\n%f\n", conj(transpose(q))*q);
}
```

Output2

```
Schur vectors q =
complex(-0.661061,0.000000) complex(-0.750332,0.000000)
```

```
complex(-0.750332,0.000000) complex(0.661061,0.000000)

Schur form t =
complex(4.594875,0.000000) complex(-2.000000,0.000000)
complex(0.000000,0.000000) complex(12.405125,0.000000)

q*t*q^H =
complex(8.000000,0.000000) complex(-3.000000,0.000000)
complex(-5.000000,0.000000) complex(9.000000,0.000000)

q^H*q =
complex(1.000000,0.000000) complex(-0.000000,0.000000)
complex(-0.000000,0.000000) complex(1.000000,0.000000)

Schur vectors q =
complex(-0.874263,0.158241) complex(-0.452441,0.076946)
complex(-0.458937,0.000000) complex(0.888419,0.009421)

Schur form t =
complex(-0.524869,1.723999) complex(-1.983020,1.403899)
complex(0.000000,0.000000) complex(10.524869,0.276001)

q*t*q^H =
complex(1.000000,2.000000) complex(-3.000000,0.000000)
complex(-5.000000,0.000000) complex(9.000000,0.000000)

q^H*q =
complex(1.000000,0.000000) complex(0.000000,-0.000000)
complex(0.000000,0.000000) complex(1.000000,0.000000)
```

See Also

eigensystem().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

sign

Synopsis
#include <**numeric.h**>
double sign(**double** *x*)

Purpose
Sign function.

Return Value
This function returns 0, 1, -1 to indicate the sign of real number *x*;

Parameters
x The input **double** type real number.

Description
The function returns the 1, -1 and 0 for $x > 0$, $x < 0$ and $x = 0$, respectively.

Example

```
#include <numeric.h>
int main() {
    int i = -10;
    float f = -10;
    double d = 10;
    double d2 = 0;
    printf("sign(i) = %d\n", sign(i));
    printf("sign(f) = %d\n", sign(f));
    printf("sign(d) = %d\n", sign(d));
    printf("sign(d2) = %d\n", sign(d2));
}
```

Output

```
sign(i) = -1
sign(f) = -1
sign(d) = 1
sign(d2) = 0
```

See Also
abs(), **real**(), **conj**().

References

sort()

Synopsis
#include <**numeric.h**>
int sort(**array double complex** &*y*, **array double complex** &*x*, ...
/* [**string_t** *method*], [**array int** &*ind*] */);

Syntax
sort(*y*, *x*)
sort(*y*, *x*, *method*)
sort(*y*, *x*, *method*, *ind*)

Purpose
Sorting and ranking elements in ascending order.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
y An array of any dimension and size which contains the sorted data.

x The same array as *x* which contains the original data.

method Sorting method.
method = ”array” - (default) sorted by total elements. That is, if *x* is an array of $n \times m$, sort in total $n \times m$ elements.
method = ”row” - sorted by row for two-dimensional array.
method = ”column” - sorted by column for two-dimensional array.

ind Index array . Each elements presents the ranking index, starting with 0, corresponding to array *x*.

Description
The original data in array *x* can be of any supported arithmetic data type and dimension. Array *y* is the same data type and size as *x* and it contains sorted data. If *x* is a complex data type, it is sorted by the magnitude of each element. If *x* includes any NaN or ComplexNaN elements, **sort**() places these at the end. The index in array *ind* contains the ranking index corresponding to array *x* and optional argument *method* specifies the sorting method. When *x* is a two-dimensional array, this parameter specifies the sorting method defined as follows: ”array” - sorted by total elements; ”row” - sorted by row for two-dimensional array; ”column” - sorted by column for two-dimensional array. By default, two-dimensional arrays are sorted by total elements. If *x* is not a two-dimensional array, The array is sorted by total elements.

Example

```
#include <numeric.h>

int main() {
    array double x1[4] = {0.1, NaN, -0.1, 3}, y1[4];
    array double complex x2[2][3] = {5, NaN, -3, -6, 3, 5}, y2[2][3];
    array int inde1[2][3];
    array double x4[3][3] = {1, 3, -3, -2, 3, 5, 9, 2, 4}, y4[3][3];
    array int inde[3][3];
    array double x3[2][3][4],y3[2][3][4];
    x3[1][1][2] = 10;
    x3[1][2][2] = -10;

    sort(y1, x1);           // sort by total elements
    printf("sort by total elements\n");
    printf("x1 = %5.2f\n", x1);
    printf("sort(x1) = %5.2f\n", y1);

    sort(y2, x2, "row", inde1);
    printf("sort by row, gives index array\n");
    printf("x2 = %f\n", x2);
    printf("sort(x2)\n");
    printf("%f\n", y2);
    printf("index(x2)\n");
    printf("%d\n", inde1);

    sort(y3, x3, "array");
    printf("sort by total elements \n");
    printf("x3 = %f\n", x3);
    printf("sort(x3)\n");
    printf("%f\n", y3);

    sort(y4, x4, "column", inde);
    printf("sort by column elements, gives index array\n");
    printf("x4 = %f\n", x4);
    printf("sort(x4)\n");
    printf("%f\n", y4);
    printf("inde(x4)\n");
    printf("%d\n", inde);
}
```

Output

```
sort by total elements
x1 =  0.10   NaN -0.10  3.00

sort(x1) = -0.10  0.10  3.00   NaN

sort by row, gives index array
x2 = complex(5.000000,0.000000) ComplexNaN complex(-3.000000,0.000000)
complex(-6.000000,0.000000) complex(3.000000,0.000000) complex(5.000000,0.000000)


sort(x2)
complex(-3.000000,0.000000) complex(5.000000,0.000000) ComplexNaN
complex(3.000000,0.000000) complex(5.000000,0.000000) complex(-6.000000,0.000000)


index(x2)
2 0 1
```

```
1 2 0

sort by total elements
x3 = 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 10.000000 0.000000
0.000000 0.000000 -10.000000 0.000000

sort(x3)
-10.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 10.000000

sort by column elements, gives index array
x4 = 1.000000 3.000000 -3.000000
-2.000000 3.000000 5.000000
9.000000 2.000000 4.000000

sort(x4)
-2.000000 2.000000 -3.000000
1.000000 3.000000 4.000000
9.000000 3.000000 5.000000

inde(x4)
1 2 0
0 0 2
2 1 1
```

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press, 1997.

specialmatrix()

Synopsis
#include <**numeric.h**>
array double specialmatrix(**string_t** *name*, ... /* [**type1** *arg1*, **type2** *arg2*, ...] */)[:][:];

Syntax
specialmatrix(*name*)
specialmatrix(*name*, *arg1*)
specialmatrix(*name*, *arg1*, *arg2*, *arg3*)

Purpose
Generate a special matrix.

Return Value
This function returns a special matrix.

Parameters
name The special matrix name selected from the table below.

arg1, *arg2*, **...** Input arguments required by a particular special matrix. The number of arguments and their data types are different for different special matrices.

Special Matrices

Cauchy	ChebyshevVandermonde	Chow	Circul	Celement
Dramadah	DenavitHartenberg	DenavitHartenberg2	Fiedler	Frank
Gear	Hadamard	Hankel	Hilbert	InverseHilbert
Magic	Pascal	Rosser	Toeplitz	Vandermonde
Wilkinson				

. . . . . .

Cauchy

Synopsis
#include <**numeric.h**>
array double specialmatrix(“Cauchy”, **array double** *c[&]*, ... /* [**array double** *r[&]*] */);

Syntax
specialmatrix(“Cauchy”, *c*)
specialmatrix(“Cauchy”, *c*, *r*)

Purpose
Generate a Cauchy matrix.

Return Value
This function returns an $n \times n$ Cauchy matrix.

Parameters
c Input vector with n elements.

c An optional input vector with n elements.

Description
This function generates the $n \times n$ Cauchy matrix whose elements are $y[i][j] = 1/(c[i] + r[j])$. When optional input *r* is not specified, then inside the function, *c* is assigned to *r*. That is, element (i, j) is defined by $y[i][j] = 1/(c[i] + c[j])$.

Example

```
#include <numeric.h>
int main() {
    array double r[5] ={3,4,8,6,2};
    array double c[5] ={8,2.3,9,6,2};
    printf("c=\n%f",c);
    printf("r=\n%f",r);
    printf("specialmatrix(\"Cauchy\", c) =\n%5.4f\n", specialmatrix("Cauchy", c));
    printf("specialmatrix(\"Cauchy\", c, r) =\n%5.4f\n", specialmatrix("Cauchy", c, r));

}
```

Output

```
c=
8.000000 2.300000 9.000000 6.000000 2.000000
r=
3.000000 4.000000 8.000000 6.000000 2.000000
specialmatrix("Cauchy", c) =
0.0625 0.0971 0.0588 0.0714 0.1000
0.0971 0.2174 0.0885 0.1205 0.2326
0.0588 0.0885 0.0556 0.0667 0.0909
0.0714 0.1205 0.0667 0.0833 0.1250
0.1000 0.2326 0.0909 0.1250 0.2500

specialmatrix("Cauchy", c, r) =
0.0909 0.0833 0.0625 0.0714 0.1000
0.1887 0.1587 0.0971 0.1205 0.2326
0.0833 0.0769 0.0588 0.0667 0.0909
0.1111 0.1000 0.0714 0.0833 0.1250
0.2000 0.1667 0.1000 0.1250 0.2500
```

References
Knuth, Donald, *The art of computer programming*, Vol. 1, Addison-Wesley Pub. Co., 1973.

ChebyshevVandermonde

Synopsis
#include <numeric.h>
array double specialmatrix(“ChebyshevVandermonde”, **array double** *c[&]*, ... /* [**int** *m*] */);

Syntax
specialmatrix(“ChebyshevVandermonde”, *c*)
specialmatrix(“ChebyshevVandermonde”, *c*, *m*)

Purpose
Generate a Vandermonde-like matrix for the Chebyshev polynomials.

Return Value
This function returns a Vandermonde-like matrix.

Parameters
c Input vector with n elements.

m An optional input integral number which defines the rows of the output Vandermonde-like matrix.

Description
This function generates the $n \times m$ Chebyshev Vandermonde matrix whose elements are $y[i][j] = T_i(P[j])$. T_{i-1} is the Chebyshev polynomial of ith degree. When the optional input *m* is not specified, the output is a square matrix of $n \times n$. If the user specifies the optional input *m*, then a rectangular version of $n \times m$ Chebyshev Vandermonde matrix is generated.

Example

```
#include <numeric.h>
int main() {
    array double v[5] = {2,3,4,5,6};
    printf("v=\n%f",v);
    printf("specialmatrix(\"ChebyshevVandermonde\", v) =\n%5.4f\n",
            specialmatrix("ChebyshevVandermonde", v));
    printf("specialmatrix(\"ChebyshevVandermonde\", v, 3) =\n%5.4f\n",
            specialmatrix("ChebyshevVandermonde", v, 3));
}
```

Output

```
v=
2.000000 3.000000 4.000000 5.000000 6.000000
specialmatrix("ChebyshevVandermonde", v) =
1.0000 1.0000 1.0000 1.0000 1.0000
```

```
2.0000 3.0000 4.0000 5.0000 6.0000
7.0000 17.0000 31.0000 49.0000 71.0000
26.0000 99.0000 244.0000 485.0000 846.0000
97.0000 577.0000 1921.0000 4801.0000 10081.0000

specialmatrix("ChebyshevVandermonde", v, 3) =
1.0000 1.0000 1.0000 1.0000 1.0000
2.0000 3.0000 4.0000 5.0000 6.0000
7.0000 17.0000 31.0000 49.0000 71.0000
```

References

N.J. Higham, *Stability analysis of algorithms for solving confluent Vandermonde-like systems*, SIAM J. Matrix Anal. Appl., 11 (1990).

Knuth, Donald Ervin, *The art of computer programming*, Vol. 1, Addison-Wesley Pub. Co., 1973.

. . . . . .

Chow

Synopsis

#include <**numeric.h**>

array double specialmatrix(“Chow”, **int** *n*, ... /* [**double** *u*, **double** *v*] */);

Syntax

specialmatrix(“Chow”, *n*)

specialmatrix(“Chow”, *n*, *u*)

specialmatrix(“Chow”, *n*, *u*, *v*)

Purpose

Generate Chow matrix.

Return Value

This function returns a Chow matrix.

Parameters

n Input integral number. It specifies the degree of the matrix. That is, the size of the Chow matrix is $n \times n$.

u An optional input value.

v An optional input value.

Description

This function generates the $n \times n$ Chow matrix. It is a singular Toeplitz lower Hessenberg matrix. This function returns a square matrix y such that $y = h + v * I$, where h is a square matrix whose elements are $h[i][j] = u^{(i-j)+1}$ and I is a $n \times n$ identity matrix. If the user does not specify the optional arguments *u* and *v*, $u = 1.0$ and $v = 0.0$ are used by default.

Example
Generate a 5th order Chow matrix with $u = 1, v = 0$(default), $u = 0.5, v = 0$ and $u = 0.5, v = 0.67$. Verify the result with the formulation $y = h + v * I$.

```
#include <numeric.h>
int main() {
    int i, j, n=5;
    double u=0.5, v=0.67;
    array double h[n][n], y[n][n];

    printf("specialmatrix(\"Chow\", n) =\n%5.4f\n", specialmatrix("Chow", n));
    printf("specialmatrix(\"Chow\", n, u) =\n%5.4f\n", specialmatrix("Chow", n, u));
    printf("specialmatrix(\"Chow\", n, u, v) =\n%5.4f\n", specialmatrix("Chow", n, u, v));
    for (i=0; i<n; i++)
       for (j=0; j<=min(i+1,n-1); j++)
          h[i][j] = pow(u,(i-j+1));
    y = h + v*identitymatrix(n);
    printf("varify matrix=\n%5.4f",y);
}
```

Output

```
specialmatrix("Chow", n) =
1.0000 1.0000 0.0000 0.0000 0.0000
1.0000 1.0000 1.0000 0.0000 0.0000
1.0000 1.0000 1.0000 1.0000 0.0000
1.0000 1.0000 1.0000 1.0000 1.0000
1.0000 1.0000 1.0000 1.0000 1.0000

specialmatrix("Chow", n, u) =
0.5000 1.0000 0.0000 0.0000 0.0000
0.2500 0.5000 1.0000 0.0000 0.0000
0.1250 0.2500 0.5000 1.0000 0.0000
0.0625 0.1250 0.2500 0.5000 1.0000
0.0312 0.0625 0.1250 0.2500 0.5000

specialmatrix("Chow", n, u, v) =
1.1700 1.0000 0.0000 0.0000 0.0000
0.2500 1.1700 1.0000 0.0000 0.0000
0.1250 0.2500 1.1700 1.0000 0.0000
0.0625 0.1250 0.2500 1.1700 1.0000
0.0312 0.0625 0.1250 0.2500 1.1700

varify matrix=
1.1700 1.0000 0.0000 0.0000 0.0000
0.2500 1.1700 1.0000 0.0000 0.0000
0.1250 0.2500 1.1700 1.0000 0.0000
0.0625 0.1250 0.2500 1.1700 1.0000
0.0312 0.0625 0.1250 0.2500 1.1700
```

References
T.S. Chow, A class of Hessenberg matrices with known eigenvalues and inverses, SIAM Review, 11 (1969), pp. 391-395.

. . . . . .

Circul

Synopsis
#include <**numeric.h**>
array double specialmatrix(“Circul”, **array double** *c[&]*);

Syntax
specialmatrix(“Circul”, *c*)

Purpose
Generate a Circul matrix.

Return Value
This function returns a $n \times n$ Circul matrix.

Parameters
c Input vector with n elements. It is the first row of Circul matrix.

Description
A circulant matrix has the property that each row is obtained from the previous one by cyclically permuting the entries one step forward. specifies the degree of the matrix.

Example

```
#include <numeric.h>
int main() {
    array double c[5] = {2,7,1,9,6};
    printf("c=\n%f",c);
    printf("specialmatrix(\"Circul\", c) =\n%5.4f\n", specialmatrix("Circul", c));
}
```

Output

```
c=
2.000000 7.000000 1.000000 9.000000 6.000000
specialmatrix("Circul", c) =
2.0000 7.0000 1.0000 9.0000 6.0000
6.0000 2.0000 7.0000 1.0000 9.0000
9.0000 6.0000 2.0000 7.0000 1.0000
1.0000 9.0000 6.0000 2.0000 7.0000
7.0000 1.0000 9.0000 6.0000 2.0000
```

References
P.J. Davis, Circulant Matrices, John Wiley, 1977.

Clement

Synopsis
#include <**numeric.h**>
array double specialmatrix(“Clement”, **int** *n* /* [**int** *k*] */);

Syntax
specialmatrix(“Clement”, *n*)
specialmatrix(“Clement”, *n*, *k*)

Purpose
Generate a Clement matrix.

Return Value
This function returns an $n \times n$ Clement matrix.

Parameters
n Input integral number. It specifies the order of the square matrix produced.

k Integral number 0 or 1. For $k = 0$ or by default, the matrix is unsymmetric. For $k = 1$, it is a symmetric matrix.

Description
Clement matrix is a tridiagonal matrix with zero diagonal elements and known eigenvalues. The eigenvalues of the matrix are positive and negative numbers of $n-1, n-3, n-5, \cdots, 1$or 0.

Example

```
#include <numeric.h>
int main() {
    int n1=5, n2=6;
    array double y1[n1][n1], eigenvalue1[n1],eigenvector1[n1][n1];
    array double y2[n2][n2], eigenvalue2[n2],eigenvector2[n2][n2];

    printf("specialmatrix(\"Clement\", n1, 0) =\n%5.4f\n",
           y1=specialmatrix("Clement", n1, 0));
    printf("specialmatrix(\"Clement\", n1, 1) =\n%5.4f",
           y1=specialmatrix("Clement", n1, 1));
    eigensystem(eigenvalue1,eigenvector1, y1);
    printf("eigenvalue=\n%f\n",eigenvalue1);
    printf("specialmatrix(\"Clement\", n2, 1) =\n%5.4f",
           y2=specialmatrix("Clement", n2, 1));
    eigensystem(eigenvalue2,eigenvector2, y2);
    printf("eigenvalue=\n%f",eigenvalue2);
}
```

Output

```
specialmatrix("Clement", n1, 0) =
0.0000 1.0000 0.0000 0.0000 0.0000
4.0000 0.0000 2.0000 0.0000 0.0000
```

```
0.0000 3.0000 0.0000 3.0000 0.0000
0.0000 0.0000 2.0000 0.0000 4.0000
0.0000 0.0000 0.0000 1.0000 0.0000

specialmatrix("Clement", n1, 1) =
0.0000 2.0000 0.0000 0.0000 0.0000
2.0000 0.0000 2.4495 0.0000 0.0000
0.0000 2.4495 0.0000 2.4495 0.0000
0.0000 0.0000 2.4495 0.0000 2.0000
0.0000 0.0000 0.0000 2.0000 0.0000
eigenvalue=
4.000000 -4.000000 0.000000 2.000000 -2.000000

specialmatrix("Clement", n2, 1) =
0.0000 2.2361 0.0000 0.0000 0.0000 0.0000
2.2361 0.0000 2.8284 0.0000 0.0000 0.0000
0.0000 2.8284 0.0000 3.0000 0.0000 0.0000
0.0000 0.0000 3.0000 0.0000 2.8284 0.0000
0.0000 0.0000 0.0000 2.8284 0.0000 2.2361
0.0000 0.0000 0.0000 0.0000 2.2361 0.0000
eigenvalue=
5.000000 -5.000000 3.000000 -3.000000 1.000000 -1.000000
```

References

P.A. Clement, A class of triple-diagonal matrices for test purposes, SIAM Review, 1 (1959), pp. 50-52.

. . . . . .

DenavitHartenberg

Synopsis

#include <**numeric.h**>

array double specialmatrix(“DenavitHartenberg”, **double** *theta* , **double** *d*, **double** *alpha*, **double** *a*);

Syntax

specialmatrix(“DenavitHartenberg”, *theta*, *d*, *alpha*, *a*)

Purpose

Generate a Denavit-Hartenberg matrix.

Return Value

This function returns a 4×4 Denavit-Hartenberg matrix.

Parameters

theta Input θ value.

d Input d value.

alpha Input α value.

a Input a value.

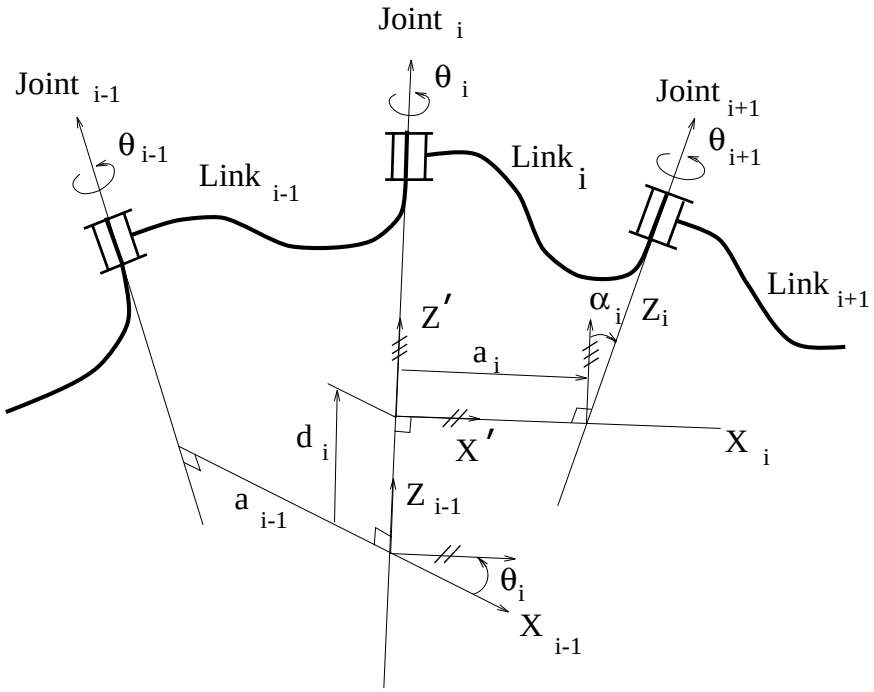


図 11.2: Link coordinate system and its parameters

Description

The Denavit-Hartenberg matrix is used to describe the coordinate transformation between adjacent links in a three dimensional space. In the Denavit-Hartenberg representation, there are three basic rules for determining and establishing the coordinate frame.

1. The z_{i-1} axis lies along the axis of motion of the ith joint.
2. The x_i axis is normal to the z_{i-1} axis, and pointing away from it.
3. The y_i axis completes the right-handed coordinate system as required.

This first Denavit-Hartenberg representation of a rigid link depends on four geometric parameters associated with each link. These four parameters completely describe any revolute or prismatic joint. Referring to the Figure11.2, these four parameters are defined as follows:

θ_i is the joint angle from the x_{i-1} axis to the x_i axis about the z_{i-1} axis (using the right-hand rule).

d_i is the distance from the origin of the $(i-1)$th coordinate frame to the intersection of the z_{i-1} axis with the x_i axis along the z_{i-1} axis.

a_i is the offset distance from the intersection of the z_i axis with the x_i axis to the origin of the ith frame along the x_i axis, or the shortest distance between the z_{i-1} and z_i axis.

α_i is the offset angle from the z_{i-1} axis to the z_i axis about the x_i axis, using the right-hand rule.

Example

```
#include<numeric.h> // with M_PI
int main() {
    array double t1[4][4], t2[4][4], t3[4][4],
                 t4[4][4], t5[4][4], t6[4][4], t[4][4];
    double d2r = M_PI/180;
    double d[6] = {0, 149.09, 0, 433.07, 0, 165.5};
    double alpha[6] = {-90*d2r, 0, 90*d2r, -90*d2r, 90*d2r, 0};
    double a[6] = {0, 431.8, -20.32, 0, 0, 0};
    double theta[6];

    theta[0] = 10*d2r;
    theta[1] = 20*d2r;
    theta[2] = 30*d2r;
    theta[3] = 40*d2r;
    theta[4] = 50*d2r;
    theta[5] = 60*d2r;
    t1 = specialmatrix("DenavitHartenberg",theta[0], d[0], alpha[0], a[0]);
    t2 = specialmatrix("DenavitHartenberg",theta[1], d[1], alpha[1], a[1]);
    t3 = specialmatrix("DenavitHartenberg",theta[2], d[2], alpha[2], a[2]);
    t4 = specialmatrix("DenavitHartenberg",theta[3], d[3], alpha[3], a[3]);
    t5 = specialmatrix("DenavitHartenberg",theta[4], d[4], alpha[4], a[4]);
    t6 = specialmatrix("DenavitHartenberg",theta[5], d[5], alpha[5], a[5]);
    t = t1*t2*t3*t4*t5*t6; // forward kinematics for Puma 560
    printf("t = \n%f\n", t);
}
```

Output

```
t =
-0.636562 0.022716 0.770891 815.135915
0.771180 0.029596 0.635929 377.870408
-0.008369 0.999304 -0.036357 140.236602
0.000000 0.000000 0.000000 1.000000
```

References

K. S. Fu, R. C. Gonzalez, C. S. G. Lee, *Robotics: Control, Seneing, Vision, and Intelligence*, McGraw-Hill Pub. Co., 1987.

. . . . . .

DenavitHartenberg2

Synopsis

#include <**numeric.h**>

array double specialmatrix(“DenavitHartenberg2”, **double** *theta* , **double** *d*, **double** *alpha*, **double** *a*);

Syntax

specialmatrix(“DenavitHartenberg2”, *theta*, *d*, *alpha*, *a*)

Purpose

generate a Denavit-Hartenberg matrix.

Return Value

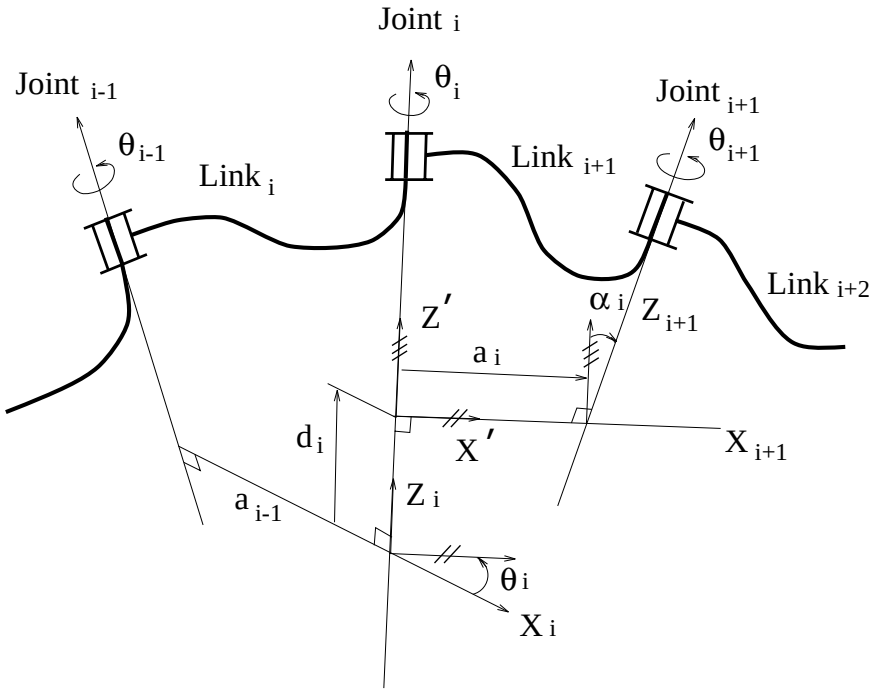


図 11.3: Link coordinate system and its parameters

This function returns a 4×4 Denavit-Hartenberg matrix.

Parameters

theta Input θ value.

d Input d value.

alpha Input α value.

a Input a value.

Description

This second Denavit-Hartenberg representation of a rigid link depends on four geometric parameters associated with each link. These four parameters completely describe any revolute or prismatic joint. Referring to the Figure11.3, these four parameters are defined as follows:

θ_i is the joint angle from the x_{i-1} axis to the x_i axis about the z_i axis, using the right-hand rule.

d_i is the distance from the origin of the $(i-1)$th coordinate frame to the intersection of the z_i axis with the x_i axis along the z_i axis.

a_i is the offset distance from the intersection of the z_i axis with the x_i axis to the origin of the $(i+1)$th frame along the x_i axis, or the shortest distance between the z_i and z_{i+1} axis.

α_i is the offset angle from the z_i axis to the z_{i+1} axis about the x_i axis, using the right-hand rule.

Example

```
#include <numeric.h>
int main() {
    printf("specialmatrix(\"DenavitHartenberg2\",M_PI, 0.0, -M_PI, 0.0) =\n%5.4f\n",
            specialmatrix("DenavitHartenberg2",M_PI, 0.0, -M_PI, 0.0));
}
```

Output

```
specialmatrix("DenavitHartenberg2",M_PI, 0.0, -M_PI, 0.0) =
-1.0000 0.0000 0.0000 0.0000
0.0000 1.0000 -0.0000 -0.0000
-0.0000 -0.0000 -1.0000 -0.0000
0.0000 0.0000 0.0000 1.0000
```

References

John J. Craig, *Robotics: Mechanics & Control*, Addison-Wesley Pub. Co., 1986.

. . . . . .

Dramadah

Synopsis

#include <**numeric.h**>

array double specialmatrix(“Dramadah1”, **int** *n* /* [**int** *k*] */);

Syntax

specialmatrix(“Dramadah1”, *n*)

specialmatrix(“Dramadah1”, *n*, *k*)

Purpose

generate a Dramadah matrix.

Return Value

This function returns an $n \times n$ Dramadah matrix.

Parameters

n Input integral number. It specifies the order of the square matrix produced.

k Integral number 1, 2, or 3.

$k = 1$ or by default, the returned matrix *y* is Toeplitz, with *abs*$|y| = 1$. Each element of y^{-1} is an integral value.

$k = 2$, *y* is upper triangular and Toeplitz. Each element of y^{-1} is an integral value.

$k = 3$, *y* has a maximal determinant among (0,1) lower Hessenberg matrices.

Description

The Dramadah matrix has elements of zeros and ones. The inverse matrix of Dramadah matrix has large integer elements.

Example

```
#include <numeric.h>
int main() {
    int n=4;
    array double y[n][n];
    printf("specialmatrix(\"Dramadah\", n) =\n%5.4f", y=specialmatrix("Dramadah", n));
    printf("absolute value of determinant of y=%f\n",abs(determinant(y)));
    printf("inverse of y=\n%5.4f\n",inverse(y));
    printf("specialmatrix(\"Dramadah\", n,2) =\n%5.4f", y=specialmatrix("Dramadah", n,2));
    printf("inverse of y=\n%5.4f\n",inverse(y));
    printf("specialmatrix(\"Dramadah\", n,3) =\n%5.4f", y=specialmatrix("Dramadah", n,3));
    printf("determinant of y=%f\n",determinant(y));
}
```

Output

```
specialmatrix("Dramadah", n) =
1.0000 1.0000 0.0000 1.0000
0.0000 1.0000 1.0000 0.0000
0.0000 0.0000 1.0000 1.0000
1.0000 0.0000 0.0000 1.0000
absolute value of determinant of y=1.000000
inverse of y=
-1.0000 1.0000 -1.0000 2.0000
1.0000 0.0000 0.0000 -1.0000
-1.0000 1.0000 0.0000 1.0000
1.0000 -1.0000 1.0000 -1.0000

specialmatrix("Dramadah", n,2) =
1.0000 1.0000 0.0000 1.0000
0.0000 1.0000 1.0000 0.0000
0.0000 0.0000 1.0000 1.0000
0.0000 0.0000 0.0000 1.0000
inverse of y=
1.0000 -1.0000 1.0000 -2.0000
0.0000 1.0000 -1.0000 1.0000
0.0000 0.0000 1.0000 -1.0000
0.0000 0.0000 0.0000 1.0000

specialmatrix("Dramadah", n,3) =
1.0000 1.0000 0.0000 0.0000
0.0000 1.0000 1.0000 0.0000
1.0000 0.0000 1.0000 1.0000
0.0000 1.0000 0.0000 1.0000
determinant of y=3.000000
```

References

R.L. Graham and N.J.A. Sloane, Anti-Hadamard matrices, Linear Algebra and Appl., 62 (1984), pp. 113-137. L. Ching, The maximum determinant of an nxn lower Hessenberg (0,1) matrix, Linear Algebra and Appl., 183 (1993), pp. 147-153.

Fiedler

Synopsis
#include <**numeric.h**>
array double specialmatrix(“Fiedler”, **array double** *c[&]*);

Syntax
specialmatrix(“Fiedler”, *c*)

Purpose
Generate a Fiedler matrix.

Return Value
This function returns an $n \times n$ Fiedler matrix.

Parameters
c Input vector with n elements. The length of the vector n specifies the order of the output matrix. That is, the size of the Fiedler matrix is $n \times n$.

Description
A Fiedler matrix has a dominant positive eigenvalue and all the other eigenvalues are negative. A Fiedler matrix is a symmetric matrix with elements *abs(c[i] - c[j])*.

Example

```
#include <numeric.h>
int main() {
    int n=5;
    array double y[n][n], eigenvalue[n], eigenvector[n][n];
    array double c[5] = {2,-7,1,-9,6};
    printf("specialmatrix(\"Fiedler\",c) =\n%5.4f", y=specialmatrix("Fiedler", c));
    eigensystem(eigenvalue,eigenvector, y);
    printf("eigenvalues =\n%f",eigenvalue);
}
```

Output

```
specialmatrix("Fiedler",c) =
0.0000 9.0000 1.0000 11.0000 4.0000
9.0000 0.0000 8.0000 2.0000 13.0000
1.0000 8.0000 0.0000 10.0000 5.0000
11.0000 2.0000 10.0000 0.0000 15.0000
4.0000 13.0000 5.0000 15.0000 0.0000
eigenvalues =
-23.319294 32.070623 -0.904663 -5.933579 -1.913088
```

References

G. Szego, Solution to problem 3705, Amer. Math. Monthly, 43 (1936), pp. 246-259.
J. Todd, Basic Numerical Mathematics, Vol. 2: Numerical Algebra, Birkhauser, Basel, and Academic Press, New York, 1977, p. 159.

. . . . . .

Frank

Synopsis

#include <**numeric.h**>
array double specialmatrix(“Frank”, **int** *n* /* [**int** *k*] */);

Syntax

specialmatrix(“Frank”, *n*)
specialmatrix(“Frank”, *n*, *k*)

Purpose

Generate a Frank matrix.

Return Value

This function returns an $n \times n$ Frank matrix.

Parameters

n Input integral number. It specifies the order of the square matrix produced.

k Integral number 0 or 1. For $k = 0$ or by default, the matrix is an upper Hessenberg with determinant 1. If $k = 1$, the elements are reflected about the anti-diagonal.

Description

A Frank matrix is an upper Hessenberg with ill-conditioned eigenvalues. The eigenvalues of the Frank matrix are positive and occur in reciprocal pairs. If n is odd, 1 is an eigenvalue. The first m, with $m =$ *floor*$(n/2)$, eigenvalues in the ascending order are ill-conditioned.

Example

```
#include <numeric.h>
int main() {
    int n1=4, n2=5;
    array double y1[n1][n1], eigenvalue1[n1], eigenvector1[n1][n1];
    array double y2[n2][n2], eigenvalue2[n2], eigenvector2[n2][n2];
    printf("specialmatrix(\"Frank\",n1) =\n%5.4f", y1=specialmatrix("Frank", n1));
    eigensystem(eigenvalue1,eigenvector1,y1);
    printf("eigenvalues =\n%5.4f",eigenvalue1);
    printf("determinant =%5.4f\n",determinant(y1));
    printf("specialmatrix(\"Frank\",n2, 1) =\n%5.4f", y2=specialmatrix("Frank", n2, 1));
    eigensystem(eigenvalue2,eigenvector2,y2);
    printf("eigenvalues =\n%5.4f",eigenvalue2);
```

```
    printf("determinant =%5.4f\n",determinant(y2));
}
```

Output

```
specialmatrix("Frank",n1) =
4.0000 3.0000 2.0000 1.0000
3.0000 3.0000 2.0000 1.0000
0.0000 2.0000 2.0000 1.0000
0.0000 0.0000 1.0000 1.0000
eigenvalues =
0.1367 7.3127 2.0666 0.4839
determinant =1.0000
specialmatrix("Frank",n2, 1) =
1.0000 1.0000 1.0000 1.0000 1.0000
1.0000 2.0000 2.0000 2.0000 2.0000
0.0000 2.0000 3.0000 3.0000 3.0000
0.0000 0.0000 3.0000 4.0000 4.0000
0.0000 0.0000 0.0000 4.0000 5.0000
eigenvalues =
3.5566 10.0629 1.0000 0.0994 0.2812
determinant =1.0000
```

References

W.L. Frank, Computing eigenvalues of complex matrices by determinant evaluation and by methods of Danilewski and Wielandt, J. Soc. Indust. Appl. Math., 6 (1958), pp. 378-392 (see pp. 385 and 388).

G.H. Golub and J.H. Wilkinson, Ill-conditioned eigensystems and the computation of the Jordan canonical form, SIAM Review, 18 (1976), pp. 578-619 (Section 13).

H. Rutishauser, On test matrices, Programmation en Mathematiques Numeriques, Editions Centre Nat. Recherche Sci., Paris, 165, 1966, pp. 349-365. Section 9.

J.H. Wilkinson, Error analysis of floating-point computation, Numer. Math., 2 (1960), pp. 319-340 (Section 8).

J.H. Wilkinson, The Algebraic Eigenvalue Problem, Oxford University Press, 1965 (pp. 92-93).

P.J. Eberlein, A note on the matrices denoted by B_n, SIAM J. Appl. Math., 20 (1971), pp. 87-92.

J.M. Varah, A generalization of the Frank matrix, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 835-839.

. . . . . .

Gear

Synopsis

#include <**numeric.h**>

array double specialmatrix(“Gear”, **int** *n*, ... /* [**int** *ni*, **int** *nj*] */);

Syntax

specialmatrix(“Gear”, *n*)

specialmatrix(“Gear”, *n*, *ni*)

specialmatrix(“Gear”, *n*, *ni*, *nj*)

Purpose
Generate a Gear matrix.

Return Value
This function returns an $n \times n$ Gear matrix.

Parameters
n Input integral number. It specifies the order of the square matrix produced.

ni Integral number *abs(ni)*$\leq n$.

nj Integral number *abs(nj)*$\leq n$.

Description
A Gear matrix is an $n \times n$ matrix with ones on the sub- and super-diagonals. The [0][abs(ni)] element is assigned the value *sign(ni)* and the [n-1][n-1-abs(nj)] is assigned *sign(nj)*. All eigenvalues of Gear matrix are of the form $2 * cos(a)$ and the eigenvectors are of the form $[sin(w+a), sin(w+2a), ..., sin(w+Na)]$, where a and w are constants.

Example

```
#include <numeric.h>
int main() {
    printf("specialmatrix(\"Gear\",7) =\n%5.4f\n", specialmatrix("Gear", 7));
    printf("specialmatrix(\"Gear\",7,3,6) =\n%5.4f\n", specialmatrix("Gear", 7, 3, 6));
}
```

Output

```
specialmatrix("Gear",7) =
0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 1.0000
1.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000
0.0000 1.0000 0.0000 1.0000 0.0000 0.0000 0.0000
0.0000 0.0000 1.0000 0.0000 1.0000 0.0000 0.0000
0.0000 0.0000 0.0000 1.0000 0.0000 1.0000 0.0000
0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 1.0000
-1.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000

specialmatrix("Gear",7,3,6) =
0.0000 1.0000 0.0000 1.0000 0.0000 0.0000 0.0000
1.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000
0.0000 1.0000 0.0000 1.0000 0.0000 0.0000 0.0000
0.0000 0.0000 1.0000 0.0000 1.0000 0.0000 0.0000
0.0000 0.0000 0.0000 1.0000 0.0000 1.0000 0.0000
0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 1.0000
1.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000
```

References
C.W. Gear, A simple set of test matrices for eigenvalue programs, Math. Comp., 23 (1969), pp. 119-125.

Hadamard

Synopsis
#include <**numeric.h**>
array double specialmatrix(“Hadamard”, **int** *n*);

Syntax
specialmatrix(“Hadamard”, *n*)

Purpose
Generate a Hadamard matrix.

Return Value
This function returns an $n \times n$ Hadamard matrix.

Parameters
n Input integral number. It specifies the degree of the matrix. That is, the size of Hadamard matrix is $n \times n$.

Description
This function generates an $n \times n$ Hadamard matrix. A Hadamard matrix is a matrix *y* with elements 1 or -1 such that $y^T * y = n * I$, where I is an $n \times n$ identity matrix. An $n \times n$ Hadamard matrix with $n > 2$ exists only if *n%4 = 0*. This function handles only the cases where $n, n/12$ or $n/20$ is a power of 2.

Example

```
#include <numeric.h>
int main() {
    int n=8;
    array double y[n][n];
    printf("specialmatrix(\"Hadamard\", n) =\n%4.1f", y=specialmatrix("Hadamard", n));
    printf("verify: transpose(y)*y-n*identitymatrix(n)=0\n%5.4f",
            transpose(y)*y-n*identitymatrix(n));
}
```

Output

```
specialmatrix("Hadamard", n) =
 1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
 1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0 -1.0
 1.0  1.0 -1.0 -1.0  1.0  1.0 -1.0 -1.0
 1.0 -1.0 -1.0  1.0  1.0 -1.0 -1.0  1.0
 1.0  1.0  1.0  1.0 -1.0 -1.0 -1.0 -1.0
 1.0 -1.0  1.0 -1.0 -1.0  1.0 -1.0  1.0
 1.0  1.0 -1.0 -1.0 -1.0 -1.0  1.0  1.0
 1.0 -1.0 -1.0  1.0 -1.0  1.0  1.0 -1.0
verify: transpose(y)*y-n*identitymatrix(n)=0
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

```
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

References

Reference: S.W. Golomb and L.D. Baumert, The search for Hadamard matrices, Amer. Math. Monthly, 70 (1963) pp. 12-17.

Hankel

Synopsis

#include <numeric.h>

array double specialmatrix(“Hankel”, **array double** *c[&]*, ... /* [**array double** *r[&]*] */);

Syntax

specialmatrix(“Hankel”, *c*)

specialmatrix(“Hankel”, *c*, *r*)

Purpose

Generate a Hankel matrix.

Return Value

This function returns a Hankel matrix.

Parameters

c Input vector with n elements.

r An optional input vector with m elements.

Description

If the optional input vector *r* is not specified, this function generates a square Hankel matrix whose column is *c* and whose elements are zero below the anti-diagonal. If the optional input vector *r* is specified, it returns an $n \times m$ Hankel matrix whose first column is *c* and last row is *r*. Hankel matrices are symmetric.

Example

```
#include <numeric.h>
int main() {
    array double r[3] ={2,6,8};
    array double c[5] ={8,2.3,9,6,2};
    printf("specialmatrix(\"Hankel\", c) =\n%5.1f\n", specialmatrix("Hankel", c));
    printf("specialmatrix(\"Hankel\", c, r) =\n%5.1f\n", specialmatrix("Hankel", c, r));
    printf("specialmatrix(\"Hankel\", r, c) =\n%5.1f\n", specialmatrix("Hankel", r, c));
```

}

Output

```
specialmatrix("Hankel", c) =
  8.0   2.3   9.0   6.0   2.0
  2.3   9.0   6.0   2.0   0.0
  9.0   6.0   2.0   0.0   0.0
  6.0   2.0   0.0   0.0   0.0
  2.0   0.0   0.0   0.0   0.0

specialmatrix("Hankel", c, r) =
  8.0   2.3   9.0
  2.3   9.0   6.0
  9.0   6.0   2.0
  6.0   2.0   6.0
  2.0   6.0   8.0

specialmatrix("Hankel", r, c) =
  2.0   6.0   8.0   2.3   9.0
  6.0   8.0   2.3   9.0   6.0
  8.0   2.3   9.0   6.0   2.0
```

References

. . . . . .

Hilbert

Synopsis
#include <**numeric.h**>
array double specialmatrix(“Hilbert”, **int** *n*);

Syntax
specialmatrix(“Hilbert”, *n*)

Purpose
Generate a Hilbert matrix.

Return Value
This function returns an $n \times n$ Hilbert matrix.

Parameters
n Input integral number. It specifies the degree of the matrix. That is, the size of Hilbert matrix is $n \times n$.

Description
This function generates an $n \times n$ Hilbert matrix. A Hilbert matrix is a matrix *y* with elements $y[i][j] = 1/(i + j + 1)$. It is a famous example of a badly conditioned matrix.

Example

```
#include <numeric.h>
int main() {
    printf("specialmatrix(\"Hilbert\", 7) =\n%5.3f\n", specialmatrix("Hilbert", 7));
}
```

Output

```
specialmatrix("Hilbert", 7) =
1.000 0.500 0.333 0.250 0.200 0.167 0.143
0.500 0.333 0.250 0.200 0.167 0.143 0.125
0.333 0.250 0.200 0.167 0.143 0.125 0.111
0.250 0.200 0.167 0.143 0.125 0.111 0.100
0.200 0.167 0.143 0.125 0.111 0.100 0.091
0.167 0.143 0.125 0.111 0.100 0.091 0.083
0.143 0.125 0.111 0.100 0.091 0.083 0.077
```

References

. . . . . .

InverseHilbert

Synopsis
#include <**numeric.h**>
array double specialmatrix(“InverseHilbert”, **int** *n*);

Syntax
specialmatrix(“InverseHilbert”, *n*)

Purpose
generate an inverse Hilbert matrix.

Return Value
This function returns an $n \times n$ inverse Hilbert matrix.

Parameters
n Input integral number. It specifies the degree of the matrix. That is, the size of the inverse Hilbert matrix is $n \times n$.

Description
This function generates an $n \times n$ inverse Hilbert matrix. The result is exact for n less than 15.

Example

```
#include <numeric.h>
int main() {
    printf("specialmatrix(\"InverseHilbert\", 7) =\n%8.4g\n",
            specialmatrix("InverseHilbert", 7));
}
```

Output

```
specialmatrix("InverseHilbert", 7) =
      49    -1176     8820 -2.94e+04 4.851e+04 -3.881e+04 1.201e+04
   -1176 3.763e+04 -3.175e+05 1.129e+06 -1.94e+06 1.597e+06 -5.045e+05
    8820 -3.175e+05 2.858e+06 -1.058e+07 1.871e+07 -1.572e+07 5.045e+06
-2.94e+04 1.129e+06 -1.058e+07 4.032e+07 -7.276e+07 6.209e+07 -2.018e+07
4.851e+04 -1.94e+06 1.871e+07 -7.276e+07 1.334e+08 -1.153e+08 3.784e+07
-3.881e+04 1.597e+06 -1.572e+07 6.209e+07 -1.153e+08 1.006e+08 -3.33e+07
1.201e+04 -5.045e+05 5.045e+06 -2.018e+07 3.784e+07 -3.33e+07 1.11e+07
```

References

. . . . . .

Magic

Synopsis
#include <**numeric.h**>
array double specialmatrix(“Magic”, **int** *n*);

Syntax
specialmatrix(“Magic”, *n*)

Purpose
generate a Magic matrix.

Return Value
This function returns an $n \times n$ Magic matrix.

Parameters
n Input integral number. It specifies the degree of the matrix. That is, the size of Magic matrix is $n \times n$.

Description
This function generates an $n \times n$ Magic matrix. A Magic matrix is a matrix *y* with elements from 1 through n^2 with equal row, column, and diagonal sums.

Example

```
#include <numeric.h>
int main() {
```

```
    int i, n=7;
    array double y[n][n], sum1[n];
    printf("specialmatrix(\"Magic\",n) =\n%4.0f", y=specialmatrix("Magic", n));
    sum(y,sum1);
    printf("Sum of columns=\n%4.0f",sum1);
    sum(transpose(y),sum1);
    printf("Sum of rows=\n%4.0f",sum1);
    printf("Sum of diagonal elements=%4.0f\n",trace(y));
    rot90(y, y);
    printf("Sum of anti-diagonal elements=%4.0f\n",trace(y));
}
```

Output

```
specialmatrix("Magic",n) =
  30   38   46    5   13   21   22
  39   47    6   14   15   23   31
  48    7    8   16   24   32   40
   1    9   17   25   33   41   49
  10   18   26   34   42   43    2
  19   27   35   36   44    3   11
  28   29   37   45    4   12   20
Sum of columns=
 175  175  175  175  175  175  175
Sum of rows=
 175  175  175  175  175  175  175
Sum of diagonal elements= 175
Sum of anti-diagonal elements= 175
```

.....

Pascal

Synopsis

#include <**numeric.h**>

array double specialmatrix(“Pascal”, **int** *n* /* [**int** *k*] */);

Syntax

specialmatrix(“Pascal”, *n*)

specialmatrix(“Pascal”, *n*, *k*)

Purpose

generate a Pascal matrix.

Return Value

This function returns an $n \times n$ Pascal matrix.

Parameters

n Input integral number. It specifies the order of the square matrix produced.

k Optional input Integral number 0, 1 or 2.

Description

If the optional input *k* is not specified by default, or specified with 0, it returns a symmetric positive definite matrix with integer elements, which is made up from Pascal's triangular. Its inverse matrix has integer elements. If the optional input *k* is specified with 1, it is a lower triangular Choleskey factor of the Pascal matrix. Its inverse matrix is itself. If the optional input *k* is specified with 2, it is a transposed and permuted version of *specialmatrix("Pascal",n,1)* which is a cubic root of the identity.

Example

```
#include <numeric.h>
int main() {
    int n=5;
    array double y[n][n];

    printf("specialmatrix(\"Pascal\", n) =\n%5.1f", y=specialmatrix("Pascal", n));
    printf("inverse matrix=\n%5.1f\n",inverse(y));
    printf("specialmatrix(\"Pascal\", n, 1) =\n%5.1f", y=specialmatrix("Pascal", n, 1));
    printf("inverse matrix=\n%5.1f\n",inverse(y));
    printf("specialmatrix(\"Pascal\", n, 2) =\n%5.1f", y=specialmatrix("Pascal", n, 2));
}
```

Output

```
specialmatrix("Pascal", n) =
  1.0   1.0   1.0   1.0   1.0
  1.0   2.0   3.0   4.0   5.0
  1.0   3.0   6.0  10.0  15.0
  1.0   4.0  10.0  20.0  35.0
  1.0   5.0  15.0  35.0  70.0
inverse matrix=
  5.0 -10.0  10.0  -5.0   1.0
-10.0  30.0 -35.0  19.0  -4.0
 10.0 -35.0  46.0 -27.0   6.0
 -5.0  19.0 -27.0  17.0  -4.0
  1.0  -4.0   6.0  -4.0   1.0

specialmatrix("Pascal", n, 1) =
  1.0   0.0   0.0   0.0   0.0
  1.0  -1.0   0.0   0.0   0.0
  1.0  -2.0   1.0   0.0   0.0
  1.0  -3.0   3.0  -1.0   0.0
  1.0  -4.0   6.0  -4.0   1.0
inverse matrix=
  1.0   0.0   0.0   0.0   0.0
  1.0  -1.0   0.0   0.0   0.0
  1.0  -2.0   1.0   0.0   0.0
  1.0  -3.0   3.0  -1.0   0.0
  1.0  -4.0   6.0  -4.0   1.0

specialmatrix("Pascal", n, 2) =
  0.0   0.0   0.0   0.0  -1.0
  0.0   0.0   0.0  -1.0   4.0
  0.0   0.0   1.0   3.0  -6.0
  0.0  -1.0  -2.0  -3.0   4.0
 -1.0  -1.0  -1.0  -1.0   1.0
```

References

. . . . . .

Rosser

Synopsis
#include <numeric.h>
array double specialmatrix("Rosser");

Syntax
specialmatrix("Rosser")

Purpose
Generate a Rosser matrix.

Return Value
This function returns an 8×8 Rosser matrix.

Description
A Rosser matrix is an 8×8 matrix *R*.

$$\begin{bmatrix} 611 & 196 & -192 & 407 & -8 & -52 & -49 & 29 \\ 196 & 899 & 113 & -192 & -71 & -43 & -8 & -44 \\ -192 & 113 & 899 & 196 & 61 & 49 & 8 & 52 \\ 407 & -192 & 196 & 611 & 8 & 44 & 59 & -23 \\ -8 & -71 & 61 & 8 & 411 & -599 & 208 & 208 \\ -52 & -43 & 49 & 44 & -599 & 411 & 208 & 208 \\ -49 & -8 & 8 & 59 & 208 & 208 & 99 & -911 \\ 29 & -44 & 52 & -23 & 208 & 208 & -911 & 99 \end{bmatrix}$$

This matrix was a challenge for many matrix eigenvalue algorithms. It has:

- Two eigenvalues with the same value.
- Three nearly equal eigenvalues.
- Dominant eigenvalues of opposite sign.
- A zero eigenvalue.
- A small, nonzero eigenvalue.

Example

```
#include <numeric.h>
int main() {
```

```
    array double y[8][8], eigenvalue[8], eigenvector[8][8];
    printf("specialmatrix(\"Rosser\") =\n%5.1f\n", y=specialmatrix("Rosser"));
    eigensystem(eigenvalue, eigenvector,y);
    printf("eigenvalue =\n%7.3f",eigenvalue);
}
```

Output

```
specialmatrix("Rosser") =
611.0 196.0 -192.0 407.0  -8.0 -52.0 -49.0  29.0
196.0 899.0 113.0 -192.0 -71.0 -43.0  -8.0 -44.0
-192.0 113.0 899.0 196.0  61.0  49.0   8.0  52.0
407.0 -192.0 196.0 611.0   8.0  44.0  59.0 -23.0
 -8.0 -71.0  61.0   8.0 411.0 -599.0 208.0 208.0
-52.0 -43.0  49.0  44.0 -599.0 411.0 208.0 208.0
-49.0  -8.0   8.0  59.0 208.0 208.0  99.0 -911.0
 29.0 -44.0  52.0 -23.0 208.0 208.0 -911.0  99.0

eigenvalue =
-1020.049  -0.000   0.098 1020.049 1020.000 1019.902 1000.000 1000.000
```

.....

Toeplitz

Synopsis
#include <**numeric.h**>
array double specialmatrix(“Toeplitz”, **array double** *c[&]*, ... /* [**array double** *r[&]*] */);

Syntax
specialmatrix(“Toeplitz”, *c*)
specialmatrix(“Toeplitz”, *c*, *r*)

Purpose
Generate a Toeplitz matrix.

Return Value
This function returns a Toeplitz matrix.

Parameters
c Input vector with n elements.

r An optional input vector with m elements.

Description
If the optional input vector *r* is not specified, this function generates a square symmetric Toeplitz matrix whose first column is *c*. If the optional input vector *r* is specified, it returns an $n \times m$ non-symmetric Toeplitz matrix whose first column is *c* and first row is *r*.

Example

```
#include <numeric.h>
int main() {
    array double r[3] ={8,6,2};
    array double c[5] ={8,2.3,9,6,2};
    printf("specialmatrix(\"Toeplitz\", c) =\n%5.1f\n", specialmatrix("Toeplitz", c));
    printf("specialmatrix(\"Toeplitz\", c, r) =\n%5.1f\n",
            specialmatrix("Toeplitz", c, r));
    printf("specialmatrix(\"Toeplitz\", r, c) =\n%5.1f\n",
            specialmatrix("Toeplitz", r, c));
}
```

Output

```
specialmatrix("Toeplitz", c) =
  8.0   2.3   9.0   6.0   2.0
  2.3   8.0   2.3   9.0   6.0
  9.0   2.3   8.0   2.3   9.0
  6.0   9.0   2.3   8.0   2.3
  2.0   6.0   9.0   2.3   8.0

specialmatrix("Toeplitz", c, r) =
  8.0   6.0   2.0
  2.3   8.0   6.0
  9.0   2.3   8.0
  6.0   9.0   2.3
  2.0   6.0   9.0

specialmatrix("Toeplitz", r, c) =
  8.0   2.3   9.0   6.0   2.0
  6.0   8.0   2.3   9.0   6.0
  2.0   6.0   8.0   2.3   9.0
```

.....

Vandermonde

Synopsis

#include <**numeric.h**>

array double specialmatrix(“Vandermonde”,**array double** *v*[&])[:][:];

syntax specialmatrix(“Vandermonde”, *v*);

Purpose

generate a Calculate the Vandermonde.

Return Value

This function returns an $n \times n$ Vandermonde matrix.

Parameters

v An input vector of size n.

Description
This function calculates the Vandermonde matrix of a vector of *n* elements. The Vandermonde matrix is an $n \times n$ matrix defined as:

$$a_{i,j} = v_i^{j-1}$$

The Vandermonde matrix can be used in the fitting of polynomial to data.

Example

```
#include <numeric.h>
int main() {
    array double v[3] = {2,3,4};
    printf("specialmatrix(\"Vandermonde\", v) =\n%f\n",
           specialmatrix("Vandermonde", v));
}
```

Output

```
specialmatrix("Vandermonde", v) =
1.000000 2.000000 4.000000
1.000000 3.000000 9.000000
1.000000 4.000000 16.000000
```

See Also
polyfit().

. . . . . .

Wilkinson

Synopsis
#include <**numeric.h**>
array double specialmatrix(“Wilkinson”, **int** *n*);

Syntax
specialmatrix(“Wilkinson”, *n*)

Purpose
generate a Wilkinson matrix.

Return Value
This function returns an $n \times n$ Wilkinson’s eigenvalue testing matrix.

Parameters
n Input integral number. It specifies the order of the square matrix produced.

Description

Wilkinson’s eigenvalue testing matrix is a symmetric, tridiagonal matrix with pairs of nearly, but not exactly, equal eigenvalues.

Example

```
#include <numeric.h>
int main() {
    int n=7;
    array double y[n][n], eigenvalue[n], eigenvector[n][n];
    printf("specialmatrix(\"Wilkinson\", n) =\n%5.1f", y=specialmatrix("Wilkinson", n));
    eigensystem(eigenvalue, eigenvector, y);
    printf("eigenvalues =\n%5.1f",eigenvalue);
}
```

Output

```
specialmatrix("Wilkinson", n) =
  3.0   1.0   0.0   0.0   0.0   0.0   0.0
  1.0   3.0   1.0   0.0   0.0   0.0   0.0
  0.0   1.0   3.0   1.0   0.0   0.0   0.0
  0.0   0.0   1.0   3.0   1.0   0.0   0.0
  0.0   0.0   0.0   1.0   3.0   1.0   0.0
  0.0   0.0   0.0   0.0   1.0   3.0   1.0
  0.0   0.0   0.0   0.0   0.0   1.0   3.0
eigenvalues =
  4.8   1.2   1.6   4.4   3.0   3.8   2.2
```

sqrtm

Synopsis
#include <**numeric.h**>
int sqrtm(**array double complex** *y[&][&]*, **array double complex** *x[&][&]*);

Syntax
sqrtm(*y*, *x*)

Purpose
Computes the square root of the matrix.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x Input square matrix. It contains data to be calculated.

y Output square matrix which contains data of the result of the principal square root of the matrix *x*.

Description
This function computes the principal square root of the matrix *x*, that is, $x = y * y$ where *x* and *y* are square matrices. If *x* has non-positive eigenvalues, the result *y* is a real matrix. If *x* has any eigenvalue of negative real part then a complex result *y* will be produced.

Example
Calculation of square roots of matrices of different data types.

```
#include <numeric.h>
int main() {
   array double x[3][3]={0.8,0.2,0.1,
                         0.2,0.7,0.3,
                         0.1,0.3,0.6};
   array double complex zx[3][3]={complex(1,1),complex(2,2),0,
                         3,complex(4,1),complex(2,5),
                         0,0,0};
   array double complex zy[3][3];
   array double y[3][3];

   sqrtm(y,x);
   printf("x = \n%5.3f",x);
   printf("y = \n%5.3f",y);
   printf("\n");

   sqrtm(zy,zx);
   printf("zx = \n%5.3f",zx);
   printf("zy = \n%5.3f",zy);

}
```

Output

```
x =
0.800 0.200 0.100
0.200 0.700 0.300
0.100 0.300 0.600
y =
0.886 0.113 0.048
0.113 0.807 0.189
0.048 0.189 0.750

zx =
complex(1.000,1.000) complex(2.000,2.000) complex(0.000,0.000)
complex(3.000,0.000) complex(4.000,1.000) complex(2.000,5.000)
complex(0.000,0.000) complex(0.000,0.000) complex(0.000,0.000)
zy =
complex(0.690,0.818) complex(1.003,0.353) complex(-3.155,0.649)
complex(1.017,-0.488) complex(1.708,0.331) complex(3.068,1.046)
complex(0.000,0.000) complex(0.000,0.000) complex(0.000,0.000)
```

See Also

logm(), **funm**(), **cfunm**(), **expm**().

References

G. H. Golub, C. F. Van Loan, Matrix Computations Third edition, The Johns Hopkins University Press, 1996

std

Synopsis
#include <**numeric.h**>
double std(**array double** &*a*, ... /* [**array double** *v*[:]] */);

Syntax
std(a)
std(a, v)

Purpose
Calculate the standard deviation of a data set.

Return Value
This function returns the standard deviation.

Parameters
a Input array which contains a data set.

v Output array which contains the standard deviations of each row of the array.

Description
This function calculates the standard deviation of a data set.

Algorithm
The standard deviation of the data set σ_i is defined as

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{j=1}^{N}(X_j - \overline{X})^2}$$

where N is the number of observations of data set *x*, $\overline{x}$ is the mean of data set *x*.

Example1

```
#include <numeric.h>
int main() {
    double a[3] = {1,2,3};
    double b[2][3] = {1,2,3,4,5,6};
    int    b1[2][3] = {1,2,3,4,5,6};
    double c[2][3][5];
    c[0][0][0] = 10;
    c[0][0][1] = 20;
    double stdval;

    stdval = std(a);
    printf("std(a) = %f\n", stdval);
    stdval = std(b);
```

```
    printf("std(b) = %f\n", stdval);
    stdval = std(b1);
    printf("std(b1) = %f\n", stdval);
    stdval = std(c);
    printf("std(c) = %f\n", stdval);
}
```

Output1

```
std(a) = 1.000000
std(b) = 1.870829
std(b1) = 1.870829
std(c) = 4.025779
```

Example2

```
#include <numeric.h>
int main() {
    double a[2][3] = {1,2,3,
                      4,5,6};
    array double b[3][4] = {1,2,3,4,
                            5,6,7,8,
                            1,2,3,4};
    array double stdv1[2], stdv2[3];

    std(a, stdv1);
    printf("std(a, stdv1) = %f\n", stdv1);
    std(b, stdv2);
    printf("std(b, stdv2) = %f\n", stdv2);
}
```

Output2

```
std(a, stdv1) = 1.000000 1.000000

std(b, stdv2) = 1.290994 1.290994 1.290994
```

See Also

mean(), **median**().

References

sum

Synopsis
#include <**numeric.h**>
double sum(**array double** &*a*, ... /*[**array double** *v*[:]]*/);

Syntax
sum(a)
sum(a, v)

Purpose
Calculate the sum of all elements of an array and the sums of each row of a two-dimensional array.

Return Value
This function returns the sum of all elements of an array.

Parameters
a The input array.

v The output array which contains the sums of each row of array.

Description
This function calculates the sum of all elements in an array of any dimension. If the array is a two-dimensional matrix, the function can calculate the sum of each row. The input array can be any arithmetic data type, and of any dimension.

Example1
Calculate the sum of all elements of arrays of different data types.

```
#include <numeric.h>
int main() {
    double a[3] = {1,2,3};
    double b[2][3] = {1,2,3,4,5,6};
    int b1[2][3] = {1,2,3,4,5,6};
    double c[2][3][5];
    c[0][0][0] = 10;
    c[0][0][1] = 20;
    double sums;

    sums = sum(a);
    printf("sum(a) = %f\n", sums);
    sums = sum(b);
    printf("sum(b) = %f\n", sums);
    sums = sum(b1);
    printf("sum(b) = %f\n", sums);
    sums = sum(c);
    printf("sum(c) = %f\n", sums);
}
```

Output1

```
sum(a) = 6.000000
sum(b) = 21.000000
sum(b) = 21.000000
sum(c) = 30.000000
```

Example2
Calculate the sums of each elements of the arrays

```
#include <numeric.h>
int main() {
    double a[2][3] = {1,2,3,
                      4,5,6};
    array double b[3][4] = {1,2,3,4,
                            5,6,7,8,
                            1,2,3,4};
    array int    b1[3][4] = {1,2,3,4,
                            5,6,7,8,
                            1,2,3,4};
    array double sumv1[2], sumv2[3];
    double s;

    s = sum(a, sumv1);
    printf("sum(a) = %f\n", sumv1);
    sum(b, sumv2);
    printf("sum(b) = %f\n", sumv2);
    sum(b1, sumv2);
    printf("sum(b1) = %f\n", sumv2);
}
```

Output2

```
sum(a) = 6.000000 15.000000

sum(b) = 10.000000 26.000000 10.000000

sum(b1) = 10.000000 26.000000 10.000000
```

See Also

csum(), **cumsum**(), cumprod(), **trace**(), **product**().

References

svd

Synopsis
#include <numeric.h>
int svd(array double complex *a*[&][&], **array double** *s*[&], **array double complex** *u*[&][&],
array double complex *vt*[&][&]);

Purpose
Compute singular value decomposition of a matrix.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
a Input matrix.

s Output S vector.

u Output U matrix.

vt Output V matrix.

Description
The function computes the singular value of a real /complex m-by-n matrix, and the left and/or right singular vectors. The SVD is formulated as

$$A = USV^T$$

Where S is an m-by-n matrix which is zero except for its min(m,n) diagonal elements, U is an m-by-m orthogonal matrix, and V is an n-by-n orthogonal matrix. The diagonal elements of S are the singular value of A; they are real, non-negative, and are in descending order. The first min(m,n) column of U and V are the left and right singular vectors of A.

Example1
Singular value decomposition of a real n-by-n matrix.

```
#include <numeric.h>
int main() {
     array double a[3][3] = {7, 8, 1,
                             3, 6, 4,
                             3, 5, 7};   /* m-by-n matrix */
     array float  b[3][3] = {7, 8, 1,
                             3, 6, 4,
                             3, 5, 7};   /* m-by-n matrix */
     int m = 3, n = 3;
     int lmn = min(m,n);
     array double s[lmn];
     array double u[m][m];
```

```
    array double vt[n][n];
    array float fs[lmn];
    array float fu[m][m];
    array float fvt[n][n];
    int status;

    svd(a, s, NULL, NULL);
    printf("s =\n%f\n", s);

    svd(a, s, u, vt);
    printf("u =\n%f\n", u);
    printf("s =\n%f\n", s);
    printf("vt =\n%f\n", vt);
    printf("u*s*v^t =\n%f\n", u*diagonalmatrix(s)*transpose(vt));

    status = svd(b, fs, fu, fvt);
    if(status ==0) {
        printf("fu =\n%f\n", fu);
        printf("fs =\n%f\n", fs);
        printf("fvt =\n%f\n", fvt);
        printf("u*s*v^t =\n%f\n", fu*diagonalmatrix(fs)*transpose(fvt));
    }
    else
        printf("error: numerical error in svd()\n");

}
```

Output1

```
s =
15.071167 5.471953 0.957939

u =
-0.660443 0.704551 0.259659
-0.511773 -0.169318 -0.842271
-0.549458 -0.689158 0.472395

s =
15.071167 5.471953 0.957939

vt =
-0.517995 0.430638 0.739075
-0.736603 0.214678 -0.641349
-0.434853 -0.876621 0.206007

u*s*v^t =
7.000000 8.000000 1.000000
3.000000 6.000000 4.000000
3.000000 5.000000 7.000000

fu =
-0.660443 0.704551 0.259659
-0.511773 -0.169318 -0.842271
-0.549458 -0.689158 0.472395

fs =
15.071167 5.471953 0.957939

fvt =
-0.517995 0.430638 0.739075
```

```
-0.736603 0.214678 -0.641349
-0.434853 -0.876621 0.206007

u*s*v^t =
7.000000 8.000000 1.000000
3.000000 6.000000 4.000000
3.000000 5.000000 7.000000
```

Example2
Singular value decomposition of a real m-by-n matrix ($m < n$).

```
#include <numeric.h>
int main() {
     array double a[2][3] = {7, 8, 1,
                             3, 6, 4}; /* m-by-n matrix */
     int m = 2, n = 3, i;
     int lmn = min(m,n);
     array double s[lmn], sm[m][n];
     array double u[m][m];
     array double vt[n][n];


     svd(a, s, u, vt);
     printf("u =\n%f\n", u);
     printf("s =\n%f\n", s);
     printf("vt =\n%f\n", vt);
     for(i=0; i<lmn; i++)
       sm[i][i] = s[i];
     printf("u*s*v^t =\n%f\n", u*sm*transpose(vt));
}
```

Output2

```
u =
-0.818910 -0.573922
-0.573922 0.818910

s =
12.851503 3.136698

vt =
-0.580020 -0.497570 0.644981
-0.777715 0.102681 -0.620174
-0.242353 0.861325 0.446525

u*s*v^t =
7.000000 8.000000 1.000000
3.000000 6.000000 4.000000
```

Example3
Singular value decomposition of a real m-by-n matrix ($m > n$).

```
#include <numeric.h>
int main() {
     array double a[3][2] = {7, 8,
                             1, 3,
                             6, 4}; /* m-by-n matrix */
     int m = 3, n = 2, i;
     int lmn = min(m,n);
```

```
     array double s[lmn], sm[m][n];
     array double u[m][m];
     array double vt[n][n];


     svd(a, s, u, vt);
     printf("u =\n%f\n", u);
     printf("s =\n%f\n", s);
     printf("vt =\n%f\n", vt);
     for(i=0; i<lmn; i++)
       sm[i][i] = s[i];
     printf("u*s*v^t =\n%f\n", u*sm*transpose(vt));
}
```

Output3

```
u =
-0.812719 -0.288579 -0.506171
-0.217573 -0.655581 0.723102
-0.540508 0.697808 0.470016

s =
13.058084 2.118123

vt =
-0.700689 0.713467
-0.713467 -0.700689

u*s*v^t =
7.000000 8.000000
1.000000 3.000000
6.000000 4.000000
```

Example4
Singular value decomposition of a real n-by-n matrix with complex number.

```
#include <numeric.h>
int main() {
     int m = 3 , n = 3;
     int lmn = min(m,n);
     array double complex a[3][3] = {7, 8, 1,
                                     3, 6, 4,
                                     3, 5, 7} ; /* m-by-n matrix */
     array complex z[3][3] = {complex(1,2), 8, 1,
                              3, 6, 4,
                              3, 5, 7} ; /* m-by-n matrix */
     array double s[lmn];
     array double complex u[m][m];
     array double complex vt[n][n];
     int status;

     svd(a, s, NULL, NULL);
     printf("s =\n%f\n", s);

     svd(a, s, u, vt);
     printf("u =\n%f\n", u);
     printf("s =\n%f\n", s);
     printf("vt =\n%f\n", vt);
     printf("u*s*v^t =\n%f\n", u*diagonalmatrix(s)*transpose(vt));
```

```
    status = svd(z, s, u, vt);
    if(status ==0) {
        printf("u =\n%f\n", u);
        printf("s =\n%f\n", s);
        printf("vt =\n%f\n", vt);
        printf("u*s*v^t =\n%f\n", u*diagonalmatrix(s)*transpose(vt));
    }
    else
        printf("error: numerical error in svd()\n");

}
```

Output4

```
s =
15.071167 5.471953 0.957939

u =
complex(-0.660443,0.000000) complex(0.704551,0.000000) complex(0.259659,0.000000)

complex(-0.511773,0.000000) complex(-0.169318,0.000000) complex(-0.842271,0.000000)

complex(-0.549458,0.000000) complex(-0.689158,0.000000) complex(0.472395,0.000000)


s =
15.071167 5.471953 0.957939

vt =
complex(-0.517995,-0.000000) complex(0.430638,-0.000000) complex(0.739075,0.000000)

complex(-0.736603,-0.000000) complex(0.214678,-0.000000) complex(-0.641349,-0.000000)

complex(-0.434853,-0.000000) complex(-0.876621,-0.000000) complex(0.206007,0.000000)


u*s*v^t =
complex(7.000000,0.000000) complex(8.000000,0.000000) complex(1.000000,0.000000)

complex(3.000000,0.000000) complex(6.000000,0.000000) complex(4.000000,0.000000)

complex(3.000000,0.000000) complex(5.000000,0.000000) complex(7.000000,0.000000)


u =
complex(-0.504577,-0.182672) complex(0.536523,-0.575026) complex(-0.247592,0.179497)

complex(-0.551746,-0.131934) complex(-0.031610,0.061460) complex(0.625115,-0.531624)

complex(-0.607942,-0.143559) complex(-0.372381,0.487898) complex(-0.372831,0.306730)


s =
13.678758 5.090474 0.989266

vt =
complex(-0.317937,-0.000000) complex(-0.358611,-0.000000) complex(0.877675,0.000000)

complex(-0.759338,-0.217182) complex(0.440159,-0.352023) complex(-0.095224,-0.222508)
```

```
complex(-0.509341,-0.125401) complex(-0.431508,0.606249) complex(-0.360819,0.202282)


u*s*v^t =
complex(1.000000,2.000000) complex(4.932809,1.184115) complex(3.850649,4.943457)

complex(3.000000,0.000000) complex(5.202331,3.116323) complex(3.380806,1.947931)

complex(3.000000,0.000000) complex(6.030656,5.110845) complex(3.373340,-0.361984)
```

Example5
Singular value decomposition of a real n-by-n singular matrix.

```
#include <numeric.h>
int main() {
    /* singular matrix */
     array float a[3][3] = {1, 2, 3,
                            4, 5, 6,
                            7, 8, 9};
     int m = 3, n = 3, i;
     int lmn = min(m,n);
     array double s[lmn], sm[m][n];
     array double u[m][m];
     array double vt[n][n];


     svd(a, s, u, vt);
     printf("u =\n%f\n", u);
     printf("s =\n%f\n", s);
     printf("vt =\n%f\n", vt);
     for(i=0; i<lmn; i++)
       sm[i][i] = s[i];
     printf("u*s*v^t =\n%f\n", u*sm*transpose(vt));
}
```

Output5

```
u =
-0.214837 0.887231 0.408248
-0.520587 0.249644 -0.816497
-0.826338 -0.387943 0.408248

s =
16.848103 1.068370 0.000000

vt =
-0.479671 -0.776691 -0.408248
-0.572368 -0.075686 0.816497
-0.665064 0.625318 -0.408248

u*s*v^t =
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000
```

See Also

condnum(), **rcondnum**().

References

E. Anderson, et al, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

trace

Synopsis
#include <**numeric.h**>
double trace(**array double** *a*[&][&]);

Purpose
Calculate the sum of diagonal elements of a matrix.

Return Value
This function returns the sum.

Parameters
a Input two-dimensional array.

Description
This function calculates the sum of the diagonal elements of the matrix.

Algorithm
For a matrix *a*, the trace is defined as

$$trace = \sum_{i=1}^{n} a_{ii}$$

Example

```
#include <numeric.h>
int main() {
    array double a[2][2] = {2, -4,
                            3, -7};
    array double a2[3][2] = {2, -4,
                             3, -7,
                             1, 1};
    array int b[2][2] = {2, -4,
                         3, -7};
    array int b2[3][2] = {2, -4,
                          3, -7,
                          1, 1};
    double t;

    t = trace(a);
    printf("trace(a) = %f\n", t);
    t = trace(a2);
    printf("trace(a2) = %f\n", t);
    t = trace(b);
    printf("trace(b) = %f\n", t);
    t = trace(b2);
    printf("trace(b2) = %f\n", t);
}
```

Output

```
trace(a) = -5.000000
trace(a2) = -5.000000
trace(b) = -5.000000
trace(b2) = -5.000000
```

See Also

ctrace(), **sum**(), **mean**(), **median**().

References

triangularmatrix

Synopsis
#include <**numeric.h**>
double triangularmatrix(**string_t** *pos*, **array double** *a*[&][&], ... /* [**int**k] */)[:][:];

Syntax
triangularmatrix(*pos*, *a*)
triangularmatrix(*pos*, *a*, *k*)

Purpose
Get the upper or lower triangular part of a matrix.

Return Value
This function returns the upper or lower triangular part of a matrix.

Parameters
pos Input string indicating which part of the matrix to be returned.

a Input 2-dimensional array.

k Input integer. A matrix is returned on and below the *k*th diagonal of the input matrix. By default, *k* is 0.

Description
This function gets a triangular matrix of matrix *a*. For *pos* of `"upper"`, the function returns the upper triangular part of matrix *a*, on and above the *k*th diagonal of the matrix. Optional argument *k* indicates the offset of the triangular matrix to the upper *k*th diagonal of the matrix. For *pos* of `"lower"` the function returns the lower part of matrix *a*, on and below the *k*th diagonal of the matrix. Optional argument *k* indicates the offset of the triangular matrix to the lower *k*th diagonal of the matrix.

Example

```
#include <numeric.h>
int main() {
    array double a[4][3] = {1,2,3,
                            4,5,6,
                            7,8,9,
                            6,3,5};
    int n = 4, m = 3, k;
    array double t[n][m];


    t = triangularmatrix("upper",a);
    printf("triangularmatrix(\"upper\", t) =\n%f\n", t);

    k = 0;
    t = triangularmatrix("upper",a,k);
    printf("triangularmatrix(\"upper\", t, 0) =\n%f\n", t);
```

```
    k = 1;
    t = triangularmatrix("upper",a,k);
    printf("triangularmatrix(\"upper\", t, 1) =\n%f\n", t);

    k = -1;
    t = triangularmatrix("upper",a,k);
    printf("triangularmatrix(\"upper\", t, -1) =\n%f\n", t);


    t = triangularmatrix("lower",a);
    printf("triangularmatrix(\"lower\", t) =\n%f\n", t);

    k = 0;
    t = triangularmatrix("lower",a,k);
    printf("triangularmatrix(\"lower\", t, 0) =\n%f\n", t);

    k = 1;
    t = triangularmatrix("lower",a, k);
    printf("triangularmatrix(\"lower\", t, 1) =\n%f\n", t);

    k = -1;
    t = triangularmatrix("lower",a,k);
    printf("triangularmatrix(\"lower\", t, -1) =\n%f\n", t);
}
```

Output

```
triangularmatrix("upper", t) =
1.000000 2.000000 3.000000
0.000000 5.000000 6.000000
0.000000 0.000000 9.000000
0.000000 0.000000 0.000000

triangularmatrix("upper", t, 0) =
1.000000 2.000000 3.000000
0.000000 5.000000 6.000000
0.000000 0.000000 9.000000
0.000000 0.000000 0.000000

triangularmatrix("upper", t, 1) =
0.000000 2.000000 3.000000
0.000000 0.000000 6.000000
0.000000 0.000000 0.000000
0.000000 0.000000 0.000000

triangularmatrix("upper", t, -1) =
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
0.000000 8.000000 9.000000
0.000000 0.000000 5.000000

triangularmatrix("lower", t) =
1.000000 0.000000 0.000000
4.000000 5.000000 0.000000
7.000000 8.000000 9.000000
6.000000 3.000000 5.000000

triangularmatrix("lower", t, 0) =
```

```
1.000000 0.000000 0.000000
4.000000 5.000000 0.000000
7.000000 8.000000 9.000000
6.000000 3.000000 5.000000

triangularmatrix("lower", t, 1) =
1.000000 2.000000 0.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000
6.000000 3.000000 5.000000

triangularmatrix("lower", t, -1) =
0.000000 0.000000 0.000000
4.000000 0.000000 0.000000
7.000000 8.000000 0.000000
6.000000 3.000000 5.000000
```

See Also

ctriangularmatrix(), **diagonal**().

unwrap

Synopsis
#include <**numeric.h**>
int unwrap(**array double** &*y*, **array double** &*x*, ... /* [**double** *cutoff*] */);

Syntax
unwrap(*y*, *x*)
unwrap(*y*, *x*, *cutoff*)

Purpose
Unwrap radian phase of each element of input array *x* by changing its absolute jump greater than π to its $2 * \pi$ complement.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x A vector or two-dimensional array which contains the original radian phase data.

y An output array which is the same dimension and size as *x*. It contains the unwraped data.

cutoff An optional input data which specifies the jump value. If the user does not specify this input, *cutoff* has a value of π by default.

Description
The input array *x* can be of a vector or a two-dimensional array. If it is a two-dimensional array, the function unwraps it through every row of the array.

Example
In motion analysis of a crank-rocker mechanism, the output range of the rocker is within $0 \sim 2\pi$. For a given mechanism, the output may be as shown on the top part in Figure 11.4. There is a jump when θ_4 is π, because $\theta_4 = \pi$ and $\theta_4 = -\pi$ are the same point for the crank-rocker mechanism. If the **unwrap**() function is used, a smooth curve for output angle θ_4 can be obtained as shown on the lower part in Figure 11.4.

```
#include <numeric.h>
#include <complex.h>
#include <chplot.h>

int main(){
    double r[1:4],theta1,theta31;
    int n1=2,n2=4, i;
    double complex z,p,rb;
    double  x1,x2,x3,x4;
    array double theta2[36],theta4[36],theta41[36];
```

```
    class CPlot subplot, *plot;

    /* four-bar linkage*/
    r[1]=5; r[2]=1.5; r[3]=3.5; r[4]=4;
    theta1=30*M_PI/180;
    linspace(theta2,0,2*M_PI);
    for (i=0;i<36;i++) {
       z=polar(r[1],theta1)-polar(r[2],theta2[i]);
       complexsolve(n1,n2,r[3],-r[4],z,x1,x2,x3,x4);

       theta4[i] = x2;
    }
    unwrap(theta41, theta4);
    subplot.subplot(2,1);
    plot = subplot.getSubplot(0,0);
    plot->data2D(theta2, theta4);
    plot->title("Wraped");
    plot->label(PLOT_AXIS_X,"Crank input: radians");
    plot->label(PLOT_AXIS_Y,"Rocker output: radians");

    plot = subplot.getSubplot(1,0);
    plot->data2D(theta2, theta41);
    plot->title("Unwraped");
    plot->label(PLOT_AXIS_X,"Crank input: radians");
    plot->label(PLOT_AXIS_Y,"Rocker output: radians");
    subplot.plotting();
}
```

Output

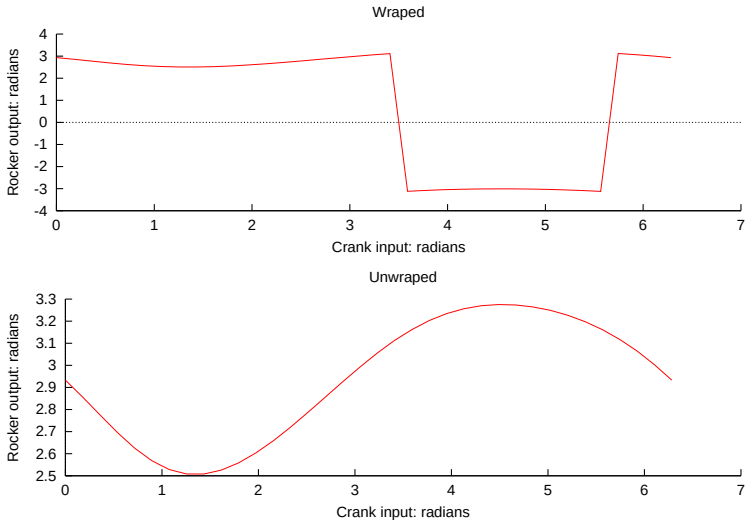


図 11.4: Comparison of results with and without using **unwrap**() function.

References

H. H. Cheng, *Computational Kinematics*, 1999.

urand

Synopsis
#include <**numeric.h**>
double urand(array double &*x*);

Syntax
urand(&*x*)
urand(*NULL*)

Purpose
Uniform random-number generator.

Return Value
This function returns a **double** uniform random-number.

Parameter
x An array of any dimensional size. It contains the uniform random numbers. The return value is the first element of array. If the argument of **urand**() is *NULL*, it only returns a uniform random number.

Description
Function urand() uses a random-number generator with period 2^{32} that returns successive pseudo-random numbers in the range from 0 to 1.

Example
The following example generates a two-dimensional array of uniform random number and a single uniform random number.

```
#include <stdio.h>
#include <numeric.h>

int main() {
   array double x[3][2];
   double u;

   u = urand(x);
   printf("urand(x) = %f\n", u);
   printf("x of urand(x) = \n%f\n", x);
   printf("urand(NULL) = %f\n", urand(NULL));
}
```

Output

```
urand(x) = 0.513871
x of urand(x) =
0.513871 0.175726
0.308634 0.534532
```

```
0.947630 0.171728

urand(NULL) = 0.702231
```

xcorr

Synopsis
#include <numeric.h>
int xcorr(array double complex *c*[&], **array double complex** *x*[&], **array double complex** *y*[&]);

Syntax
xcorr(*c*, *x*, *y*)

Purpose
One-dimensional cross-correlation function estimation.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

c A one-dimensional **array** of size $(n+m-1)$. It contains the result of a cross-correlation function estimate of *x* and *y*.

x A one-dimensional **array** of size n. It contains data used for correlation.

y A one-dimensional **array** of size m. It contains data used for correlation.

Description
The input **array** *x* and *y* can be of any supported arithmetic data type and size. Conversion of the data to **double complex** is performed internally. If both *x* and *y* are **real** data, the result is a one-dimensional **real array** *c* of size $(n + m - 1)$. If either one of *x* or *y* is **complex** , then the result **array** *c* will be of **complex** data type.

Algorithm
Given two functions $x(t)$ and $y(t)$, and their corresponding Fourier transforms $X(f)$ and $Y(f)$, the cross correlation of these two functions, denoted as $xcorr(x, y)$, is defined by

$$xcorr(x, y) \equiv \int_{-\infty}^{\infty} x(t + \tau)y(t)d\tau$$

$$xcorr(x, y) \iff X(f)Y^*(f)$$

The asterisk denotes a complex conjugate. The correlation of a function with itself is called its autocorrelation. In this case

$$xcorr(x, x) \iff |X(f)^2|$$

In discrete correlation of two sampled sequences x_k and y_k, each periodic with period N, is defined by

$$xcorr(x,y)_j \equiv \sum_{k=0}^{N-1} x_{j+k} y_k$$

The discrete correlation theorem says that this discrete correlation of two real sequences x and y is one member of the discrete Fourier transform pair

$$xcorr(x,y)_j \quad \Longleftrightarrow \quad X_k Y_k^*$$

where X_k and Y_k^* are discrete Fourier transforms of x_k and y_k, and the asterisk denotes complex conjugation. This theorem makes the same presumptions about the functions as those encountered for the discrete convolution theorem. The correlation can be computed using the DFT as follows: DFT the two data sequences, multiplying one resulting transform by the complex conjugate of the other, and inverse transforming the product, the result is the correlation. Just as in the case of convolution, time-aliasing is handled by padding with zeroes.
In the numerical implementation, the sizes of two data sequences, **array** *x* of size n and *y* of size m, are expanded to $(m+n-1)$ and padded with zeroes internally. The FFT algorithm is used to compute the discrete Fourier transforms of *x* and *y* first. Multiplying the complex conjugate of transform of *y* with the transform of *x* component by component. Then using the inverse FFT algorithm to take the inverse transform of the products, the result is the cross correlation $xcorr(x,y)$ for arrays *x* and y.

Example
For two sequences *x*[m] and *y*[n], analytically, the cross correlation of *x* and *y* can be calculated by

$$c[k] = \sum_{j=\max\{1,k-n+1\}}^{\min\{k,m\}} y[j]x[n+j-k]; \qquad\qquad k = 1, 2, \cdots, n+m-1$$

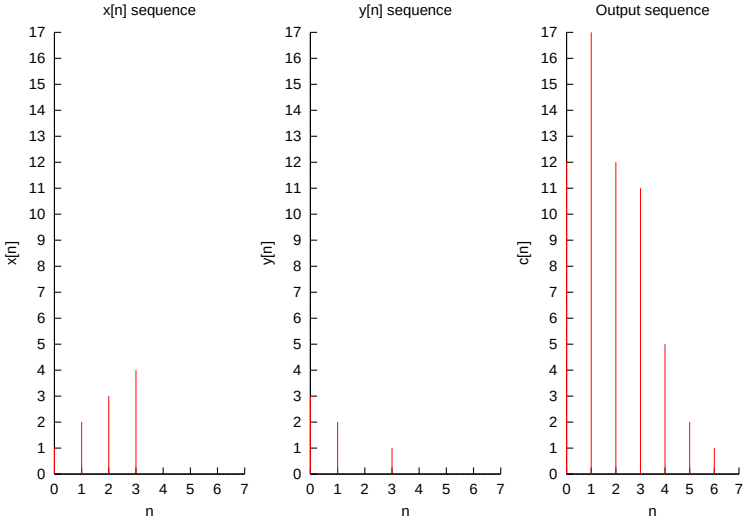
Given $x = \{1,2,3,4\}$ and $y = \{3,2,0,1\}$, then

$$\begin{aligned}
c[1] &= y[1]x[4] = 12 \\
c[2] &= y[1]x[3] + y[2]x[4] = 17 \\
c[3] &= y[1]x[2] + y[2]x[3] + y[3]x[4] = 12 \\
c[4] &= y[1]x[1] + y[2]x[2] + y[3]x[3] + y[4]x[4] = 11 \\
c[5] &= y[2]x[1] + y[3]x[2] + y[4]x[3] = 5 \\
c[6] &= y[3]x[1] + y[4]x[2] = 2 \\
c[7] &= y[4]x[1] = 1
\end{aligned}$$

Output

```
x= 1.000  2.000  3.000  4.000
```

```
y= 3.000  2.000  0.000  1.000

c=12.000 17.000 12.000 11.000  5.000  2.000  1.000
```



See Also

fft(), **ifft**(), **conv**(), **conv**2(), **deconv**().

References

William H. Press, et al, *Numerical Recipes in C*, second edition, Cambridge University Press, 1997.

第12章　非ローカルジャンプ <setjmp.h>

ヘッダー **setjmp.h** は、通常の関数呼び出しと戻り規則をバイパスするために、マクロ **setjmp** を定義し、1 つの関数と 1 つの型を宣言します。

次の型が宣言されます。

jmp_buf
これは、呼び出し元環境に戻るために必要な情報の保持に適した配列型です。

次のマクロが定義されます。
setjmp

これは、現在の環境を保存します。

関数

setjmp.h ヘッダーファイルには次の関数が定義されています。

関数	説明
longjmp	**setjmp** マクロの最新の呼び出しによって保存された環境に戻ります。

マクロ

setjmp.h ヘッダーファイルには次のマクロが定義されています。

マクロ	説明
setjmp	呼び出し元環境を **jmp_buf** 引数に保存します。

宣言型

setjmp.h ヘッダーファイルには次の関数が定義されています。

型	説明

jmp_buf	array - 呼び出し元環境に戻るために必要な情報を保持します。

移植性

このヘッダーには移植性に関する既知の問題はありません。

setjmp

構文

#include <**setjmp.h**>
int setjmp(jmp_buf *env*);

目的

環境を保存します。

戻り値

直接的な呼び出しから戻る場合、 **setjmp** マクロは値ゼロを返します。 **longjmp** 関数の呼び出しから戻る場合、**setjmp** マクロはゼロ以外の値を返します。**setjmp** マクロの呼び出しは、次のコンテキストのいずれかでのみ実行する必要があります。

– 選択ステートメントまたは繰り返しステートメントの制御式全体
– 関係演算子または等値演算子のオペランドで、他方のオペランドが整数の定数式であり、結果式が選択ステートメントまたは繰り返しステートメントの制御式全体である

– 単項「!」演算子のオペランドで、結果式が選択ステートメントまたは繰り返しステートメントの制御式全体である

– 式ステートメントの式全体 (場合によっては **void** にキャストされます)

パラメータ

env 呼び出し元環境を保存するための **jmp_buf** 型変数。

説明

setjmp マクロは、後で **longjmp** 関数から使用するために、呼び出し元環境を **jmp_buf** 引数に保存します。

例

```
#include <stdio.h>
#include <stdlib.h>
#include <setjmp.h>

jmp_buf env;
int i = 0;
main ()
{
    void exit();

    if(setjmp(env) != 0) {
       int k;

       printf("value of i on 2nd return from setjmp: %d\n", i);
       i = 2;
```

```
            k = setjmp(env);
            printf("k = %d\n", k);
            if(k==0) {
              printf("reset setjmp() \n");
              restore_env();
            }
            else {
              printf("value of i on 3rd return from setjmp: %d\n", i);
              exit(0);
            }
        }
        (void) printf("value of i on 1st return from setjmp: %d\n", i);
        i = 1;
        restore_env();
        /* NOTREACHED */
    }
    int restore_env()
    {
        longjmp(env, 1);
        /* NOTREACHED */
        return 0;
    }
```

出力

```
value of i on 1st return from setjmp: 0
value of i on 2nd return from setjmp: 1
k = 0
reset setjmp()
k = 1
value of i on 3rd return from setjmp: 2
```

関連項目

longjmp().

longjmp

構文

#include <**setjmp.h**>
void longjmp(**jmp_buf** *env*, **int** *val*);

目的

環境に戻ります。

戻り値

longjmp が完了すると、プログラムの実行は、対応する **setjmp** マクロの呼び出しにより *val* が指定した値が返されたのと同じように続行されます。 **longjmp** 関数では、 **setjmp** マクロが値 0 を返すようにはできません。*val* が 0 の場合、 **setjmp** マクロは値 1 を返します。

パラメータ

env マクロで呼び出し元環境が保存された **jmp_buf** 型変数。

val 関数で呼び出し元に返される値。0 を指定することはできません。

説明

longjmp 関数は、対応する **jmp_buf** 引数を使用した同じプログラム呼び出し内で、**setjmp** マクロが最後に呼び出されたときに保存された環境を復元します。このような呼び出しがなかった場合、**setjmp** マクロの呼び出しを含む関数の実行がその時点までに終了していた場合、または **setjmp** マクロの呼び出しが任意に変更可能な型を持つ識別子のスコープ内にあり、その時点までの実行でそのスコープ外になった場合、動作は不定です。

すべてのアクセス可能なオブジェクトの値は、 **longjmp** が呼び出された時点の値になります。ただし、 **setjmp** の呼び出しと **longjmp** の呼び出しの間に変更された volatile 修飾型を持たない **setjmp** マクロをこの関数が含んでいる場合は、ローカルな自動記憶期間のオブジェクトの値は不定です。

例

setjmp() を参照してください。

関連項目

setjmp().

第13章　シグナル処理 <signal.h>

ヘッダー **signal.h** は、さまざまなシグナル (プログラムの実行中に報告される可能性がある状態) を処理するために、1 つの型と 2 つの関数を宣言し、さまざまなマクロを定義します。

次の型が定義されます。

sig_atomic_t

これは、非同期割り込みが存在する場合でもアトミックエンティティとしてアクセスできるオブジェクトの整数型です (場合によっては volatile 修飾されます)。

次のマクロが定義されます。

SIG_DFL
SIG_ERR
SIG_IGN

これらのマクロは、**signal** 関数への 2 番目の引数および **signal** 関数の戻り値と互換性がある型の個別の値を持つ定数式に展開され、それらの値は、宣言可能なすべての関数のアドレスとの不一致を比較します。次のシグナルは、個別のシグナル番号の値を持つ **int** 型の正の整数定数式に展開され、それぞれ、特定の状況に該当します。

SIGABRT 異常終了。abort 関数によって開始された場合など
SIGFPE 算術演算エラー。ゼロによる除算や結果がオーバーフローする演算など

SIGILL 無効な関数イメージの検出。無効な命令など

SIGINT 対話型注意シグナルの受信

SIGSEGV 記憶域への無効なアクセス

SIGTERM プログラムに送信された終了要求

Ch では、これらのシグナルのいずれも生成することはありません。ただし、 **raise** 関数を明示的に呼び出した場合は除きます。文字 **SIG** と大文字で始まる追加のシグナル及び、**SIG_** と大文字で始まるマクロ定義を使用した宣言できない関数への追加のポインタも異なるプラットフォームの実装ごとに指定することができます。シグナルの完全セット、意味、および既定の処理は、プラットフォームに依存します。すべてのシグナル番号は、正の数でなければなりません。

関数

signal.h ヘッダーファイルには次の関数が定義されています。

関数	説明
signal()	シグナル番号 sig の処理方法を決定します。

マクロ

signal.h ヘッダーファイルでは、次のマクロと **SIG** で始まるその他のさまざまなマクロを定義します。

マクロ	説明
SIG_DFL	既定のシグナル処理の使用を指定します。
SIG_ERR	関数 **signal()** でエラーが発生したときの値を返します。
SIG_IGN	信号を無視します。

宣言型

signal.h ヘッダーファイルでは、次の型を定義します。The following types are defined by the **signal**.h header file.

型	説明
sig_atomic_t	int - アトミックエンティティとしてアクセスできるオブジェクト

移植性

このヘッダーには移植性に関する既知の問題はありません。

signal

構文
#include <signal.h>
void (*signal(int *sig*, **void (*** *func*)(**int**)))(**int**);

目的
シグナルの受信方法を選択します。

戻り値
要求が承認される場合、 **signal** 関数は、指定されたシグナル *sig* による **signal** の呼び出しが最後に成功したときの *func* 値を返します。それ以外の場合は **SIG_ERR** の値が返され、 **errno** に正の値が格納されます。

パラメータ
sig シグナル番号

func シグナルハンドラへのポインタ、または定義済み定数 **SIG_DFL** または **SIG_IGN**。

説明
signal 関数は、シグナル番号 sig を受信して処理するための方法を 3 つの中から 1 つ選択します。*func* の値が **SIG_DFL** の場合は、そのシグナルに対する既定の処理が発生します。*func* の値が **SIG_IGN** の場合、シグナルは無視されます。それ以外の場合、*func* は、そのシグナルが発生したときに呼び出される関数をポイントする必要があります。シグナルによる関数の呼び出し、またはその呼び出しによって呼び出される (再帰的な) 任意の関数 (標準ライブラリ以外の関数) はシグナルハンドラと呼ばれます。

シグナルの発生時に *func* が関数を参照している場合、**signal**(*sig*, **SIG_DFL**); に相当する処理を実行するかどうか、または、現在のシグナル処理が完了するまで実装で定義されたシグナルセット (少なくとも *sig* を含む) の発生を禁止するかどうかは、実装で定義されます。**SIGILL** の場合、アクションを実行しないことを実装で定義できます。(**func*)(sig); に相当する処理が実行されます。関数が戻るときに、*sig* の値が **SIGFPE** 、**SIGILL** 、 **SIGSEGV** 、または計算上の例外に対応する実装で定義されたそれ以外の値の場合、動作は不定です。それ以外の場合、プログラムは、割り込みが発生した時点から実行を再開します。

abort 関数または **raise** 関数を呼び出した結果シグナルが発生している場合、シグナルハンドラは、**raise** 関数を呼び出してはなりません。

シグナルの発生が **abort** 関数または **raise** 関数の呼び出し結果ではない場合、シグナルハンドラが **volatile sig_atomic_t** として宣言されたオブジェクトへの値の割り当て以外の静的な記憶期間を持つ任意のオブジェクトを参照するときは、動作は不定です。また、シグナルハンドラが **abort** 関数または

最初の引数がハンドラの呼び出しを発生させたシグナルに対応するシグナル番号を持った **signal** 関数以外の標準ライブラリの任意の関数を呼び出す場合も、動作は不定です。さらに、 **signal** 関数へのこのような呼び出しによって **SIG_ERR** が返される場合、**SIG_ERR** の値は不定です。

プログラムの起動時に、いくつかのシグナルに対して、
signal(sig, SIG_IGN);
に相当する処理が、実装で定義された方法で実行されます。
signal(sig, SIG_DFL);
に相当する処理は、実装で定義されたそれ以外のすべてのシグナルに対して実行されます。

実装では、ライブラリ関数による **signal** 関数の呼び出しが存在しないかのように動作する必要があります。

例

```
/* sample program to show how signal() is used */
#include <stdio.h>
#include <signal.h>
int delay = 1;
void childHandler();

int main()
{
   int pid;
   char *argv[4];

   argv[0]=strdup("ls");
   argv[1]=strdup("-s");
   argv[2]=strdup("signal.c");
   argv[3]=NULL;
   signal(SIGCHLD, childHandler);
   pid=fork();
   if(pid==0)
   {
      execvp("ls",argv);
      perror("limit");
   }
   else
   {
      sleep(delay);
      printf("Child %d exceeded limit and is being killed \n",pid);
      kill(pid, SIGINT);
   }
}

void childHandler()
{
   int childPid, childStatus;
   childPid=wait(&childStatus);
   printf("Child %d terminated within %d seconds \n", childPid, delay);
   exit(0);
}
```

出力

```
   2 signal.c
Child 9211 terminated within 1 seconds
```

関連項目

raise().

raise

構文

#include <**signal.h**>
int raise(**int** *sig*);

目的

sig で指定されたシグナルを実行中のプログラムに送信します。

戻り値

raise 関数は、成功した場合はゼロを返し、失敗した場合はゼロ以外の値を返します。

パラメータ

sig シグナル番号

説明

raise 関数は、signal 関数によって記述されたアクションを実行します。シグナルハンドラが呼び出された場合、シグナルハンドラが戻った後でなければ **raise** 関数が戻ることはできません。

例

```
#include<stdio.h>
#include<signal.h>

void func(int sig) {
   printf("sig in func() = %d\n", sig);
}

int main() {
  void (*signal (int sig, void (*disp)(int)))(int);
  void (*fptr)(int);

  fptr = signal(SIGTERM, func);
  printf("fptr = %p\n", fptr);
  raise(SIGTERM);
}
```

出力

```
fptr = 0
sig in func() = 15
```

関連項目

signal().

第14章　可変長引数リスト — <stdarg.h>

ヘッダー **stdarg.h** には、1 つの型が宣言され、8 つのマクロが定義されています。これらは、解析時に呼び出し先関数に個数と型が認識されない引数のリストを処理します。

これにより、引数の個数と型をさまざまに変えて関数を呼び出すことができます。パラメータリストには 1 つ以上のパラメータが含まれます。一番右のパラメータ (ここでは *parmN* で表す) は、アクセスメカニズムで特殊な役割を果たします。

宣言型 **va_list** は、**va_start** 、**va_arg** 、**va_end** 、**va_copy** 、**va_count** 、**va_datatype** 、**va_arraydim** 、**va_arrayextent** 、**va_arraynum** 、**va_arraytype** 、**va_tagname** の各マクロに必要な情報を保持するのに適したオブジェクト型です。可変長引数にアクセスする場合、呼び出し先関数では **va_list** 型のオブジェクト (ここでは *ap* と呼ぶ) を宣言する必要があります。オブジェクト *ap* は、引数として別の関数に渡すことができます。関数がパラメータ *ap* を指定して **va_arg** マクロを呼び出す場合、呼び出し元関数の *ap* の値は不定であるため、*ap* をさらに参照する前に **va_end** マクロに *ap* を渡す必要があります。

可変長引数リストのアクセスマクロ

ここで説明している **va_start** 、 **va_arg** 、および **va_copy** マクロは、関数ではなくマクロとして実装されています。**va_end** は外部リンケージで宣言される識別子です。プログラムで **va_end** という名前の外部識別子を定義する場合、その動作は不定です。**va_start** マクロまたは **va_copy** マクロの各呼び出しは、可変個の引数を受け取る関数内の対応する **va_end** マクロの呼び出しと一致している必要があります。

マクロ

stdarg.h ヘッダーファイルには次のマクロが定義されています。

マクロ	説明
VA_NOARG	**va_start**() の 2 番目の引数 (関数に渡す引数がない場合)。
va_arg	呼び出し元の関数内に指定された型と次の引数の値を含む式に展開されます。
va_arraydim	可変長引数の配列次元を取得します。
va_arrayextent	可変長引数の配列次元の要素数を取得します。
va_arraynum	可変長引数の配列内の要素数を取得します。
va_arraytype	可変長引数の配列の型を取得します。
va_copy	**va_list** のコピーを作成します。

va_count	可変長引数の数を取得します。
va_datatype	可変長引数のデータ型を取得します。
va_dim	非推奨。代わりに **va_arraydim** を使用します。
va_elementtype	非推奨。代わりに **va_datatype** を使用します。
va_end	関数から正常復帰させます。
va_extent	非推奨。代わりに **va_arrayextent** を使用します。
va_start	以降に **va_arg** および **va_end***ap* を初期化します。
va_tagname	可変長引数の struct/class/union 型引数のタグ名を取得します。.

宣言される型

stdarg.h ヘッダーファイルには次の型が定義されています。

型	説明
va_list	オブジェクト ― プレフィックス **va_**が付いた関数に必要な情報を保持します。

マクロ

ヘッダーファイル **ch.h** 内に定義されたデータ型 **ChType_t** には次の値があります。

値	説明
CH_UNDEFINETYPE	未定義型。
CH_CHARTYPE	char 型。
CH_UCHARTYPE	unsigned char 方。
CH_SHORTTYPE	short int 型。
CH_USHORTTYPE	unsigned short int 型。
CH_INTTYPE	int/long int 型。
CH_UINTTYPE	unsigned int 型。
CH_LLINTTYPE	long long int 型。
CH_ULLINTTYPE	unsigned long long int 型。
CH_FLOATTYPE	float 型。
CH_DOUBLETYPE	double 型。
CH_LDOUBLETYPE	long double 型。
CH_COMPLEXTYPE	float complex 型。
CH_LCOMPLEXTYPE	long complex 型。
CH_STRINGTYPE	string_t 型。
CH_FILETYPE	FILE 型。
CH_VOIDTYPE	void 型。

CH_PROCTYPE	function 型。
CH_STRUCTTYPE	struct 型。
CH_CLASSTYPE	class 型。
CH_UNIONTYPE	union 型。
CH_ENUMTYPE	enum 型。
CH_CARRAYTYPE	C 配列型。
CH_CARRAYPTRTYPE	c 配列へのポインタ。
CH_CARRAYVLATYPE	C VLA 配列。
CH_CHARRAYTYPE	Ch 配列型。
CH_CHARRAYPTRTYPE	cH 配列へのポインタ。
CH_CHARRAYVLATYPE	Ch VLA 配列。

CH_NULLTYPE	NULL ポインタ型
CH_VOIDPTRTYPE	void 型へのポインタ。
CH_CHARPTRTYPE	char 型へのポインタ。
CH_UCHARPTRTYPE	unsigned char 型へのポインタ。
CH_SHORTPTRTYPE	short int 型へのポインタ。
CH_USHORTPTRTYPE	unsigned short int 型へのポインタ。
CH_INTPTRTYPE	int/long int 型へのポインタ。
CH_UINTPTRTYPE	unsigned int 型へのポインタ。
CH_LLINTPTRTYPE	long long int 型へのポインタ。
CH_ULLINTPTRTYPE	unsigned long long int 型へのポインタ。
CH_FLOATPTRTYPE	float 型へポインタ。
CH_DOUBLEPTRTYPE	double 型へのポインタ。
CH_LDOUBLEPTRTYPE	long double 型へのポインタ。
CH_COMPLEXPTRTYPE	float complex 型へのポインタ。
CH_LCOMPLEXPTRTYPE	long complex 型へのポインタ。
CH_STRINGPTRTYPE	string_t 型へのポインタ。
CH_PROCPTRTYPE	関数ポインタ型。
CH_FILEPTRTYPE	FILE 型へのポインタ。
CH_STRUCTPTRTYPE	struct 型へのポインタ。
CH_CLASSPTRTYPE	class 型へのポインタ。
CH_UNIONPTRTYPE	union 型へのポインタ。
CH_ENUMPTRTYPE	enum 型へのポインタ。

CH_VOIDPTR2TYPE	void 型のポインタへのポインタ。
CH_CHARPTR2TYPE	char 型のポインタへのポインタ。
CH_UCHARPTR2TYPE	unsigned char 型のポインタへのポインタ。
CH_SHORTPTR2TYPE	short int 型のポインタへのポインタ。
CH_USHORTPTR2TYPE	unsigned short int 型のポインタへのポインタ。
CH_INTPTR2TYPE	int/long int 型のポインタへのポインタ。
CH_UINTPTR2TYPE	unsigned int 型のポインタへのポインタ。

CH_LLINTPTR2TYPE	long long int 型のポインタへのポインタ。
CH_ULLINTPTR2TYPE	unsigned long long int 型のポインタへのポインタ。
CH_FLOATPTR2TYPE	float 型のポインタへのポインタ。
CH_DOUBLEPTR2TYPE	double 型のポインタへのポインタ。
CH_LDOUBLEPTR2TYPE	long double 型のポインタへのポインタ。
CH_COMPLEXPTR2TYPE	float complex 型のポインタへのポインタ。
CH_LCOMPLEXPTR2TYPE	long complex 型のポインタへのポインタ。
CH_STRINGPTR2TYPE	string_t 型のポインタへのポインタ。
CH_FILEPTR2TYPE	FILE 型のポインタへのポインタ。
CH_STRUCTPTR2TYPE	struct 型のポインタへのポインタ。
CH_CLASSPTR2TYPE	class 型のポインタへのポインタ。
CH_UNIONPTR2TYPE	union 型のポインタへのポインタ。
CH_ENUMPTR2TYPE	enum 型のポインタへのポインタ。

関数

関数	説明
arraycopy()	有効な各種の型の配列をコピーします。

移植性

このヘッダーには移植性に関する既知の問題はありません。

arraycopy

構文

#include <**stdarg.h**>
int arraycopy(void **des*, **ChType_t** *destype*, **void** **src*, **ChType_t** *srctype*, **int** *n*);

目的

有効な各種の型の *n* 個の要素の配列を *src* から *des* にコピーします。

戻り値

関数が成功した場合は 0、それ以外の場合はゼロ以外の値を返します。

パラメータ

des コピー先配列のアドレス。

destype コピー先配列のデータ型。有効なデータ型はヘッダーファイル **stdarg.h** で定義されます。

src コピー元配列のアドレス。

srctype コピー元配列のデータ型。有効なデータ型はヘッダーファイル **stdarg.h** で定義されます。

n 配列の要素数。

説明

関数 **arraycopy()** は、*src* に指定されたアドレスおよびデータ型から *des* に指定されたアドレスおよびデータ型に配列をコピーします。コピー先配列とコピー元配列は、データ型は異なっていてもかまいませんが、要素の個数は同じであることを前提としています。可変長引数から渡される配列のデータ型は、関数 **va_datatype()** によって識別できます。データ型 `destype` または `srctype` がポインタ型の場合、対応する非ポインタ型として処理されます。

例

```
#include <numeric.h>

int main () {
    int i, a[6], *p;
    array double b[6];

    lindata(2, 12, a);
    printf("a = ");
    for(i=0; i<6; i++) {
      printf("%d ", a[i]);
    }
    p = &a[0];
    lindata(20, 120, p, 6);
    printf("\na = ");
    for(i=0; i<6; i++) {
      printf("%d ", a[i]);
    }
```

```
    printf("\nb = ");
    lindata(2, 12, b);
    printf("%g", b);
}
```

出力

```
a = 2 4 6 8 10 12
a = 20 40 60 80 100 120
b = 2 4 6 8 10 12
```

関連項目

va_start(), **va_arg**(), **va_count**(), **va_datatype**(), **va_arraytype**().

va_arg

構文

#include <**stdarg.h**>
type **va_arg(va_list** *ap*, *type*);

目的

可変長引数リストの値を展開します。

戻り値

va_start マクロ呼び出し後の最初の **va_arg** 呼び出しでは、*parmN* で指定された引数の直後の引数の値を返します。続けて実行すると、残りの引数の値を順番に返します。

パラメータ

ap **va_list** 型のオブジェクト。

type 渡された可変長引数のデータ型。

説明

va_arg マクロは、呼び出しで指定された次の引数の型と値を持つ式に展開されます。パラメータ *ap* は、**va_start** によって初期化された **va_list** の *ap* と同じでなければなりません。**va_arg** を呼び出すたびに *ap* は変更され、次回の呼び出しの際には、さらに次の引数を返します。パラメータ *type* の型名は、*type* の後に*****を付けるだけで、指定された型を持つオブジェクトへのポインタの型が取得できるように指定する必要があります。次の実引数がない場合や、*type* に次の実引数の (既定の上位変換に応じて引数を変換した) 型との互換性がない場合、動作は不定です。

例

va_start() を参照してください。

関連項目

va_start(), **va_end**().

va_arraydim

構文
#include <**stdarg.h**>
int va_arraydim(va_list *ap*);

目的
可変長引数の配列次元を取得します。

戻り値
va_arraydim マクロは配列の次元を返します。

パラメータ
ap **va_list** 型のオブジェクト。

説明
va_arraydim マクロは、可変長引数の配列の次元を返します。

例
va_start() を参照してください。

関連項目
va_count(), **va_datatype**(), **va_arrayextent**(), **va_arraynum**(), **va_arraytype**().

va_arrayextent

構文

#include <**stdarg.h**>
int va_arrayextent(va_list *ap*, **int** *i*);

目的

可変長引数の指定した次元の要素数を取得します。

戻り値

va_arrayextent マクロは、可変長引数の指定した次元の要素数を返します。

パラメータ

ap **va_list** 型のオブジェクト。

i 要素数を取得する次元を指定する整数。

説明

va_arrayextent マクロは、可変長引数の指定した次元の要素数を返します。

例

va_start() を参照してください。

関連項目

va_count(), **va_datatype**(), **va_arraydim**(), **va_arraynum**(), **va_arraytype**().

va_arraynum

構文

#include <**stdarg.h**>
int va_arraynum(**va_list** *ap*);

目的

可変長引数から渡された配列の要素数を取得します。

戻り値

va_arraynum マクロは、可変長引数から渡された配列の要素数を返します。渡された引数が配列型ではない場合、0 を返します。

パラメータ

ap **va_list** 型のオブジェクト。

説明

va_arraynum マクロは、可変長引数から渡された配列の要素数を返します。

例

va_start() を参照してください。

関連項目

va_count(), **va_datatype**(), **va_arraydim**(), **va_arrayextent**(), **va_arraytype**().

va_arraytype

構文

#include <**stdarg.h**>
ChType_t va_arraytype(va_list *ap*);

目的

可変長引数から渡された引数が配列であるかどうか、およびその配列型を判断します。

戻り値

引数に基づいて、この関数はヘッダーファイル **stdarg.h** に定義された次のいずれかのマクロを返します。

マクロ	説明	例
CH_UNDEFINETYPE	配列でない	`int i`
CH_CARRAYTYPE	C 配列	`int a[3]`
CH_CARRAYPTRTYPE	C 配列へのポインタ	`int (*ap)[3]`
CH_CARRAYVLATYPE	C VLA 配列	`int a[n]` `int func(int a[n], int b[:], int c[&])`
CH_CHARRAYTYPE	Ch 配列	`array int a[3]`
CH_CHARRAYPTRTYPE	Ch 配列へのポインタ	`array int (*ap)[3]`
CH_CHARRAYVLATYPE	Ch VLA 配列	`array int a[n]; int fun(array int a[n],` `array int b[:], array int c[&])`

パラメータ

ap **va_list** 型のオブジェクト。

説明

関数 **va_arraytype()** は、渡された引数が配列型であるかどうかを判断します。配列のデータ型は、関数 **va_datatype()** によって識別する必要があります。

例

va_start() を参照してください。

```
#include <stdarg.h>
#include <array.h>

void fun (int k, ...) {
    int i, ii, vacount;
    va_list ap;
    char *c;
    int *a;
    double (*b)[6];

    va_start(ap, k);
    vacount = va_count(ap);
```

```
    for(i = 0; i<vacount; i++) {
      if(va_arraytype(ap)==CH_CARRAYTYPE) {
         if(va_datatype(ap) == CH_CHARTYPE) {
           c = va_arg(ap, char *);
           printf("c = %s\n", c);
         }
         else if(va_datatype(ap) == CH_INTTYPE) {
           a = va_arg(ap, int *);
           printf("a[0] = %d\n", a[0]);
         }
      }
      else if(va_arraytype(ap)==CH_CHARRAYTYPE) {
         b = (double (*)[6])va_arg(ap, double *);
         printf("b[0][0] = %f\n", b[0][0]);
      }
      else
         ii = va_arg(ap, int);
    }
    va_end(ap);
}

int main() {
   int i;
   char c[10] = "hello";
   int a[4] = {8, 2, 3, 4};
   array double b[5][6] = { 10, 2};

   fun(i, c);
   fun(i, a);
   fun(i, b, a, 10);
}
```

出力

```
c = hello
a[0] = 8
b[0][0] = 10.000000
a[0] = 8
```

関連項目

va_count(), **va_datatype**(), **va_arraydim**(), **va_arrayextent**(), **va_arraynum**().

va_copy

構文

#include <**stdarg.h**>
void va_copy(va_list *dest*, **va_list** *src*);

目的

va_list *src* を **va_list** *dest* にコピーします。

戻り値

va_copy マクロは値を返しません。

パラメータ

dest コピー先オブジェクト。

src コピー元オブジェクト。

説明

va_copy マクロは、以前に *src* の現在の状態を取得するために使用した **va_arg** マクロと同じ順序で **va_list** マクロが適用されたものとして、**va_list** *src* のコピー **va_list** *dest* を作成します。

例

```
#include <stdarg.h>

void fun (int i, ...) {
    int i, j, k;;
    float f;
    va_list ap, ap2;

    va_start(ap, i);
    j = va_arg(ap, int);
    printf("j = %d\n", j);
    va_copy(ap2, ap);

    k = va_arg(ap, int);
    printf("k = %d\n", k);
    f = va_arg(ap, float);
    printf("f = %f\n", f);
    va_end(ap);

    /* use ap2 copied from ap */
    k = va_arg(ap2, int);
    printf("k = %d\n", k);
    f = va_arg(ap2, float);
    printf("f = %f\n", f);
    va_end(ap2);
}
```

```
int main() {
   fun(1, 2, 3, 4.0);
}
```

出力

```
j = 2
k = 3
f = 4.000000
k = 3
f = 4.000000
```

関連項目

va_arg(), **va_start**(), **va_end**().

va_count

構文
#include <**stdarg.h**>
int va_count(**va_list** *ap*);

目的
可変長引数の数を取得します。

戻り値
va_count マクロは、可変長引数の数を返します。

パラメータ
ap **va_list** 型のオブジェクト。

説明
va_count マクロは、可変長引数の数を返します。

例
va_start() を参照してください。

関連項目
va_dim(), **va_extent**(), **va_elementtype**().

va_datatype

構文
#include <**stdarg.h**>
ChType_t va_datatype(**va_list** *ap*);

目的
可変長引数のデータ型を取得します。

戻り値
va_datatype マクロは、可変長引数のデータ型を返します。

パラメータ
ap **va_list** 型のオブジェクト。

説明
va_datatype マクロは、可変長引数のデータ型を返します。渡された引数が配列型の場合、配列の要素の型を返します。

例
va_start() を参照してください。
関連項目
va_count(), **va_arraytype**(),

va_dim

構文
#include <**stdarg.h**>
int va_dim(va_list *ap*);

目的
可変長引数の配列次元を取得します。

戻り値
va_dim マクロは配列の次元を返します。

パラメータ
ap **va_list** 型のオブジェクト。

説明
va_dim マクロは、可変長引数の配列の次元を返します。

注
va_dim() は非推奨です。**va_arraydim()** を使用してください。

例
va_start() を参照してください。

関連項目
va_arraydim().

va_elementtype

構文
#include <**stdarg.h**>
int va_elementtype(**va_list** *ap*);

目的
可変長引数のデータ型を取得します。

戻り値
va_elementtype マクロは、可変長引数のデータ型を返します。

パラメータ
ap **va_list** 型のオブジェクト。

説明
va_elementtype マクロは、可変長引数のデータ型を返します。

注
va_elementtype() は非推奨です。**va_datatype**() を使用してください。

例
va_start() を参照してください。

関連項目
va_count(), **va_dim**(), **va_extent**().

va_end

構文
#include <**stdarg.h**>
void va_end(va_list *ap*);

目的
関数から正常復帰します。

戻り値
va_end マクロは値を返しません。

パラメータ
ap **va_list** 型のオブジェクト。

説明
va_end マクロは、 **va_list** *ap* を初期化した **va_start** の展開で参照された可変長引数リストを持つ関数からの正常な復帰を可能にします。 **va_end** マクロは *ap* を変更し、途中で **va_start** を呼び出さないと使用できないようにします。対応する **va_start** マクロの呼び出しがない場合、または復帰の前に **va_end** マクロが呼び出されない場合、動作は不定です。
例
va_start() を参照してください。

関連項目
va_start(), **va_arg()**.

va_extent

構文

#include <**stdarg.h**>
int va_extent(**va_list** *ap*, **int** *i*);

目的

可変長引数の指定した次元の要素数を取得します。

戻り値

va_extent マクロは、可変長引数の指定した次元の要素数を返します。

パラメータ

ap **va_list** 型のオブジェクト。

i 要素数を取得する次元を指定する整数。

説明

va_extent マクロは、可変長引数の指定した次元の要素数を返します。

注

va_extent() は非推奨です。**va_arrayextent**() を使用してください。

例

va_start() を参照してください。

関連項目

va_arrayextent().

va_start

構文

#include <**stdarg.h**>
void va_start(va_list *ap*, *parmN*);

目的

ap を初期化します。

戻り値

va_start マクロは値を返しません。

パラメータ

ap **va_list** 型のオブジェクト。

parmN 関数定義内の可変長パラメータリスト内の一番右のパラメータの識別子。

説明

va_start マクロは、名前のない引数にアクセスする前に呼び出す必要があります。
以降に行われる **va_arg** および **va_end()** の呼び出しに備えて *ap* を初期化します。同じ *ap* で再度 **va_start** を呼び出す場合は、その前に **va_end** を呼び出す必要があります。
パラメータ *parmN* は、関数定義中の可変長パラメータリストの一番右のパラメータの識別子 (省略記号,... の直前の識別子) です。パラメータ *parmN* は、**register** 記憶クラス、関数型または配列型、または、既定の上位変換を引数に適用した結果の型と互換性がない型として宣言できます。関数にパラメータがない場合は VA_NOARG を使用する必要があります。

例

```
#include <stdarg.h>
#include <array.h>

void fun (int k, ...) {
    int m, n, vacount, num;
    va_list ap;

    va_start(ap, k);
    vacount = va_count(ap);
    printf("va_count(ap) = %d\n", vacount);
    if(va_arraytype(ap)==CH_CARRAYTYPE ||
       va_arraytype(ap)==CH_CHARRAYTYPE) {
      printf("va_arraydim(ap)= %d\n", va_arraydim(ap));
      num = va_arraynum(ap);
      printf("va_arraynum(ap)= %d\n", num);
      m = va_arrayextent(ap, 0);
      printf("va_arrayextent(ap, 0)= %d\n", m);
      if(va_arraydim(ap) > 1) {
        n = va_arrayextent(ap, 1);
        printf("va_arrayextent(ap, 1)= %d\n", n);
```

```
      }
      if(va_datatype(ap) == CH_INTTYPE) {
        int a[num], *p;
        ChType_t dtype;
        void *vptr;
        dtype = va_datatype(ap);
        vptr = va_arg(ap, void *);
        printf("array element is int\n");
        p = vptr;
        printf("p[0] = %d\n", p[0]);
        arraycopy(a, CH_INTTYPE, vptr, dtype, num);
        printf("a[0] = %d\n", a[0]);
      }
      else if(va_datatype(ap) == CH_DOUBLETYPE) {
        array double b[m][n];
        ChType_t dtype;
        void *vptr;
        dtype = va_datatype(ap);
        vptr = va_arg(ap, void *);
        printf("array element is double\n");
        arraycopy(&b[0][0], CH_DOUBLETYPE, vptr, dtype, num);
        printf("b = \n%f", b);
      }
    }
    else  if(va_datatype(ap) == CH_INTTYPE)
      printf("data type is int\n");
    else  if(va_datatype(ap) == CH_DOUBLETYPE)
      printf("data type is double\n");
    else  if(va_datatype(ap) == CH_INTPTRTYPE)
      printf("data type is pointer to int\n");
    va_end(ap);
}

int main() {
   int i, a[4]={10, 20, 30}, *p;
   double b[2][3]={1, 2, 3, 4, 5, 6};

   p = &i;
   fun(i, a);
   fun(i, b,a);
   fun(i, p);
}
```

出力

```
va_count(ap) = 1
va_arraydim(ap)= 1
va_arraynum(ap)= 4
va_arrayextent(ap, 0)= 4
array element is int
p[0] = 10
a[0] = 10
va_count(ap) = 2
va_arraydim(ap)= 2
va_arraynum(ap)= 6
va_arrayextent(ap, 0)= 2
va_arrayextent(ap, 1)= 3
array element is double
b =
1.000000 2.000000 3.000000
```

```
4.000000 5.000000 6.000000
va_count(ap) = 1
data type is pointer to int
```

関連項目

va_arg(), **va_end**().

va_tagname

構文

#include <**stdarg.h**>
char va_tagname(**va_list** *ap*);

目的

可変長引数の struct/class/union 型のタグ名を取得します。

戻り値

va_tagname マクロは char 型のポインタのタグ名を返します。引数の型が struct、class、union のいずれでもない場合、NULL を返します。

パラメータ

ap **va_list** 型のオブジェクト。

説明

va_tagname マクロは char 型のポインタのタグ名を返します。引数の型が struct、class、union のいずれでもない場合、NULL を返します。

例

プログラム va_tagname.ch

```
#include<stdio.h>
#include<stdarg.h>

struct tag1 {int i; int j;};
class tag2  {public: int i; int j;};
union tag3  {int i; double d;};

void f1(int i, ...) {
  va_list ap;
  struct tag1 s, *sp;
  class tag2 c, *cp;
  union tag3 u, *up;

  va_start(ap, arg_num);
  if(va_datatype(ap) == CH_STRUCTTYPE) {
    if(!strcmp(va_tagname(ap), "tag1"))
      s = va_arg(ap, struct tag1);
  }
  if(va_datatype(ap) == CH_STRUCTPTRTYPE) {
    if(!strcmp(va_tagname(ap), "tag1"))
      sp = va_arg(ap, struct tag1*);
  }

  if(va_datatype(ap) == CH_CLASSTYPE) {
    if(!strcmp(va_tagname(ap), "tag2"))
```

```
      c = va_arg(ap, class tag2);
  }
  if(va_datatype(ap) == CH_CLASSPTRTYPE) {
    if(!strcmp(va_tagname(ap), "tag2"))
      cp = va_arg(ap, class tag2*);
  }

  if(va_datatype(ap) == CH_UNIONTYPE) {
    if(!strcmp(va_tagname(ap), "tag3"))
      u = va_arg(ap, union tag3);
  }
  if(va_datatype(ap) == CH_UNIONPTRTYPE) {
    if(!strcmp(va_tagname(ap), "tag3"))
      up = va_arg(ap, union tag3*);
  }

  printf("i =  %d\n", i);
  printf("s.i = %d\n", s.i);
  printf("s.j = %d\n", s.j);
  printf("sp->i = %d\n", sp->i);
  printf("sp->j = %d\n", sp->j);
  printf("c.i = %d\n", c.i);
  printf("c.j = %d\n", c.j);
  printf("cp->i = %d\n", cp->i);
  printf("cp->j = %d\n", cp->j);
  printf("u.d = %f\n", u.d);
  printf("up->d = %f\n", up->d);
  va_end(ap);
}

int main(){
  struct tag1 s = {1,2};
  class tag2 c = {3,4};
  union tag3 u;
  u.d = 5;

  f1(10, s, &s, c, &c, u, &u);
}
```

出力

```
i =  10
s.i = 1
s.j = 2
sp->i = 1
sp->j = 2
c.i = 3
c.j = 4
cp->i = 3
cp->j = 4
u.d = 5.000000
up->d = 5.000000
```

関連項目

va_datatype().

第15章　ブール型とその値 <stdbool.h>

ヘッダー **stdbool.h** は、4 つのマクロを定義しています。

マクロ
　bool
は **_Bool** に展開されます。

残りの 3 つのマクロは**#if** プリプロセッサディレクティブ内での使用に適しています。これらは、それぞれ以下のとおりです。
　true
は整数定数 1 に展開され、
　false
は整数定数 0 に展開され、そして
　__bool_true_false_are_defined
は 10 進定数 1 に展開されます。

予約済みの識別子であるにもかかわらず、プログラム中でマクロ **bool、true、** および **false** を非定義にしてから再定義することは許されています。

マクロ
ヘッダーファイル **stdbool.h** では、以下のマクロが定義されています。

マクロ	説明
bool	**_Bool** に展開されます。
true	整数定数 1 に展開されます。
false	整数定数 0 に展開されます。
__bool_true_false_are_defined	10 進定数 1 に展開されます。

第16章　共通定義 <stddef.h>

標準ヘッダー **stddef.h** には、次の型とマクロが定義されています。一部は他のヘッダーにも定義されていますが、これについては該当する項で説明します。

次の型が定義されています。

ptrdiff_t

2 つのポインタの差を表す符号付き整数型です。

size_t

sizeof 演算子の結果を表す符号なし整数型です。

wchar_t

サポートされているロケールの中で指定された最大の拡張文字セットのすべてのメンバに対して、異なるコードを表現できる値の範囲を持つ整数型です。null 文字はコード値が 0 である必要があります。基本文字セットの各メンバは、整数文字定数で単一の文字として使用された場合の値に等しいコード値を持つ必要があります。

定義されているマクロは次のとおりです。

offsetof(*type, member-designator***)**

size_t 型を持つ整数の定数式に展開されます。その値は、type で指定した構造体の先頭から member-designator で指定した構造体メンバまでのバイト単位のオフセットです。type (型) と member designator (メンバ指示子) は、type で指定した構造体の変数を

static *type* **t;**

で定義した場合、式**&(t.member-designator)** がアドレス定数に評価されるようにする必要があります (指定されたメンバがビットフィールドの場合、動作は未定義です)。

マクロ

stddef.h ヘッダーファイルには次のマクロが定義されています。

マクロ	説明
offsetof	整数定数に展開されます。

宣言される型

stddef.h ヘッダーファイルには次の型が定義されています。

型	説明
ptrdiff_t	int - 2 つのポインタの差
size_t	unsigned int - **sizeof** 演算子の結果
wchar_t	int - 値の範囲は、サポートされているロケールの中で指定された最大の拡張文字セットのすべてのメンバに対して、異なるコードを表現できます。

移植性

このヘッダーには移植性に関する既知の問題はありません。

第17章　入出力 <stdio.h>

ヘッダー **stdio.h** には入出力を実行するための 3 つの型、複数のマクロ、および多数の関数が宣言されています。

宣言されている型は、いずれも **size_t** (**stddef.h** で説明) です。

file

この型は、ファイル位置インジケータ、関連付けられているバッファへのポインタ (ある場合)、読み取り/書き込みエラーが発生したかどうかを記録するエラーインジケータ、ファイルの終わりに到達したかどうかを記録する *EOF (end-of-file)* インジケータなど、ストリームの制御に必要なすべての情報を記録できるオブジェクト型です。

fpos_t

この型は、配列型以外で、ファイル内のすべての位置を一意に指定するために必要なすべての情報を記録できるオブジェクト型です。

定義されているマクロは次のとおりです。

_IOFBF

_IOLBF

_IONBF

これらのマクロは異なる値を持つ整数の定数式に展開されます。値は **setvbuf** 関数の第 3 引数としての使用に適しています。

BUFSIZ

このマクロは整数の定数式に展開され、setbuf 関数で使用するバッファのサイズを定義します。

EOF

このマクロは、 **int** 型で負の値を持つ整数の定数式に展開されます。これはストリームからそれ以上の入力がないこと、つまり、ファイルの終わりを示すために複数の関数によって返されます。

FOPEN_MAX

このマクロは整数の定数式に展開され、実装で保証されている同時に開くことが可能なファイルの最大の個数を定義します。

FILENAME_MAX

このマクロは整数の定数式に展開され、実装で保証されている開くことが可能な最長のファイル名文字列を格納するために必要な配列 **char** のサイズを定義します。

L_tmpnam

このマクロは整数の定数式に展開され、**tmpnam** 関数が生成する一時ファイル名文字列を格納するために必要な配列 char のサイズを定義します。

SEEK_CUR

SEEK_END
SEEK_SET
これらのマクロは異なる値を持つ整数の定数式に展開されます。値は **fseek** 関数の第 3 引数としての使用に適しています。

TMP_MAX
このマクロは整数の定数式に展開され、 **tmpnam** 関数で生成できる一意のファイル名の最大の個数を定義します。

stderr
stdin
stdout
これらのマクロは、標準エラーストリーム、入力ストリーム、および出力ストリームにそれぞれ関連付けられたファイルオブジェクトを指す " **file** へのポインタ"型の式です。

ヘッダー **wchar.h** には、ワイド文字の入出力に有用な多数の関数が宣言されています。その副文節に記述されているワイド文字入出力関数は、ここで説明する大部分の処理と同様の処理を実行します。ただし、プログラム内部の基本単位がワイド文字であるという点が異なります。外部表現 (ファイル内) は、"汎用化された"マルチバイト文字のシーケンスです。

入出力関数とは、次の関数の総称です。

– ワイド文字入力関数– 記述された、ワイド文字およびワイド文字列への入力を実行する次の関数: **fgetwc, fgetws, getwc, getwchar, fwscanf, wscanf, vfwscanf, vwscanf**
– ワイド文字出力関数 – ワイド文字およびワイド文字列からの出力を実行する関数: **fputwc, fputws, putwc, putwchar, fwprintf, wprintf, vfwprintf, vwprintf**

– ワイド文字入出力関数– **ungetwc** 関数、ワイド文字入力関数、およびワイド文字出力関数の和集合

– バイトの入出力関数– この副文節に記述された、入出力を実行する関数: **fgetc, fgets, fprintf, fputc, fputs, fread, fscanf, fwrite, getc, getchar, gets, printf, putc, putchar, puts, scanf, ungetc, vfprintf, vfscanf, vprintf, vscanf**

ストリーム
端末やテープドライブなどの物理デバイスからの入出力や構造化された記憶装置に格納されたファイルからの入出力など、すべての入出力は論理データストリームにマッピングされます。このストリームの特性は、さまざまな種類の入力や出力にかかわらず一貫しています。テキストストリームとバイナリストリームという 2 つのマッピング形式がサポートされます。

テキストストリームは、行を構成する順序付けられた文字のシーケンスです。各行はゼロ個以上の文字と行末の改行文字で構成されています。最後の行に終端の改行文字が必要であるかどうかはプラットフォームに依存します。ホスト環境でテキストを表現するためのさまざまな規則に準拠するために、

入力および出力に文字を追加、変更、または削除することが必要な場合があります。このため、ストリーム内の文字と外部表現の文字とが 1 対 1 で対応しているとは限りません。テキストストリームから読み込まれたデータは、次の場合に限り、以前にストリームに書き出されたデータと必ず等しくなります。すなわち、データが出力文字および制御文字 (水平タブおよび改行) のみで構成されていて、空白文字の直前に改行文字がなく、最後の文字が改行文字である場合です。改行文字の直前に書き出された空白文字が読み込み時に出力されるかどうかは、プラットフォームに依存します。

バイナリストリームも順序付けられている文字のシーケンスですが、内部データを透過的に記録することができます。バイナリストリームから読み込まれたデータは、同じ実装下で以前にそのストリームに書き出されたデータと等しくなければなりません。ただし、このようなストリームでは、プラットフォームに依存する文字数の null 文字がストリームの終わりに追加される場合があります。

各ストリームには指向性があります。ストリームを外部ファイルに関連付けた後、ストリーム上で操作を実行するまでは、ストリームは無指向の状態です。ワイド文字の入出力関数を無指向ストリームに適用すると、ストリームはワイド指向ストリームになります。同様に、バイトの入出力関数を無指向ストリームに適用すると、ストリームはバイト指向ストリームになります。ストリームの指向性を変更できるその他の方法は、**freopen** 関数または **fwide** 関数の呼び出しだけです。(**freopen** への関数呼び出しに成功すると、指向性はなくなります。)
バイトの入出力関数はワイド指向ストリームに適用するべきではありません。また、ワイド文字の入出力関数をバイト指向ストリームに適用するべきではありません。その他のストリーム操作はストリームの指向性に影響を与えることも、ストリームの指向性に影響を受けることもありませんが、以下の追加の制限があります。

– バイナリのワイド指向ストリームには、テキストストリームおよびバイナリストリームの両方に適用されるファイルの位置決めに関する制限があります。

– ワイド指向ストリームでは、ファイル位置決め関数の呼び出しが正常に終了してファイル位置インジケータがファイルの終わりよりも前にある状態になった後に、ワイド文字出力関数が 1 文字のマルチバイト文字の一部分を上書きする場合があります。上書きされたバイト以降のファイルの内容は不定になります。

ワイド指向の各ストリームには関連付けられた **mbstate_t** オブジェクトがあり、そこにストリームの現在の解析状態が格納されています。**fgetpos** への関数呼び出しに成功した場合は、この **mbstate_t** オブジェクトの値を **fpos_t** オブジェクトの値の一部として示す表現が格納されます。格納されている **fpos_t** の値と同じ値を使用して **fsetpos** への関数呼び出しをもう一度行い、その呼び出しに成功すると、関連付けられている **mbstate_t** オブジェクトの値と制御ストリーム内での位置が復元されます。

環境の制限

各プラットフォームは、終端の改行文字を含めて少なくとも 254 文字を含む行で構成されるテキストファイルをサポートしています。マクロ **BUFSIZ** の値は少なくとも 256 です。

ファイル

ストリームを外部ファイル (物理デバイスなど) に関連付けるには、ファイルを開きます。ファイル

を開くことで新しいファイルを作成する場合もあります。作成するファイルが既に存在する場合は、必要に応じて前の内容が破棄されます。ファイルが位置決め要求 (端末ではなくディスクファイルなど) をサポートできる場合、ストリームに関連付けられたファイル位置インジケータはファイルの先頭 (文字番号ゼロ) に位置付けられます。ただし、ファイルが書き込みモードで開かれている場合は例外です。その場合には、ファイル位置インジケータの最初の位置がファイルの先頭になるか終わりになるかはプラットフォームに依存します。ファイル位置インジケータは、後続の読み取り、書き込み、および位置決め要求によって保持され、ファイルの先頭から末尾まで処理が正しく進行します。

後述する例外を除き、バイナリファイルは切り詰められません。テキストストリームに対する書き込みにより、関連付けられたファイルでそのポイント以降が切り詰められるかどうかは、プラットフォームに依存します。
ストリームをバッファリングしない場合、文字は可能な限り迅速にソースから取り出されるか、ターゲットに置かれます。そうでない場合は、文字を蓄積し、ブロックにまとめてホスト環境との間で送受信します。ストリームをフルバッファリングする場合、バッファがいっぱいになったときに文字をブロックにまとめてホスト環境との間で送受信します。ストリームをラインバッファリングする場合は、改行文字が出現するたびに、ホスト環境との間で文字をブロックにまとめて送受信します。また、バッファがいっぱいになったとき、バッファリングしないストリームに対して入力が要求されたとき、またはホスト環境からの文字の送信を要求するラインバッファリングされたストリームに対して入力が要求されたときは、文字をブロックにまとめてホスト環境との間で送受信します。これらの特性のサポートはプラットフォームに依存します。また、 **setbuf** 関数および **setvbuf** 関数により影響を受ける場合があります。
ファイルを閉じると、ファイルと制御ストリームの関連付けを解除できます。ストリームとファイルの関連付けが解除される前に、出力ストリームがフラッシュされます (書き込まれていないバッファの内容はホスト環境に送信されます)。関連付けられたファイル (標準テキストストリームを含む) を閉じた後のファイルオブジェクトへのポインタの値は不定です。長さゼロのファイル (出力ストリームによって文字が書き込まれていない) が実際に存在するかどうかは、プラットフォームに依存します。

ファイルは、同一または別のプログラム実行によって後で再度開くことができます。また、その内容を再利用または変更できます (ファイルの先頭に再位置付けできる場合)。**main** 関数が元の呼び出し元に戻るか、**exit** 関数が呼び出されると、開いているすべてのファイルが閉じ (したがってすべての出力ストリームがフラッシュされ)、プログラムが終了します。**abort** 関数の呼び出しなど、その他の方法でのプログラム終了の場合は、すべてのファイルが正常に閉じられるとは限りません。

ストリームの制御に使用する FILE オブジェクトのアドレスが重要である場合があります。FILE オブジェクトのコピーが元の FILE オブジェクトの代わりの役割を果たすとは限りません。

プログラムの起動時、標準入力 (従来の入力の読み込み用)、標準出力 (従来の出力の書き込み用)、および標準エラー (診断出力書き込み用) の 3 つのテキストストリームは事前に定義されており、明示的に開く必要はありません。最初に開いた時点では、標準エラーストリームはフルバッファリングされません。標準入力ストリームおよび標準出力ストリームは、ストリームが対話型デバイスを参照しないことが確認できた場合に限り、フルバッファリングされます。

追加 (一時でない) ファイルを開く関数にはファイル名が必要です。有効なファイル名を構成するための規則はプラットフォームに依存します。同一ファイルを複数回同時に開くことができるかどうかもプラットフォーム依存です。

テキストおよびバイナリのワイド指向ストリームは両方とも、概念的にはワイド文字のシーケンスですが、ワイド指向ストリームに関連付けられている外部ファイルはマルチバイト文字のシーケンスであり、次のように汎用化されます。

– ファイル内のマルチバイトエンコーディングは、埋め込みの null バイトを含むことができます (プログラム内部での使用に有効なマルチバイトエンコーディングとは異なります)。

– ファイルは初期シフト状態で開始または終了するとは限りません。

さらに、マルチバイト文字で使用されるエンコーディングは、ファイルごとに異なる場合があります。このようなエンコーディングの性質および選択はいずれも、プラットフォームに依存します。

ワイド文字入力関数は、 **fgetwc** 関数の連続呼び出しによってマルチバイト文字が読み込まれた場合と同様に、ストリームからマルチバイト文字を読み込んでワイド文字に変換します。個々の変換は **mbrtowc** 関数の呼び出しによる変換と同様に行われます。変換状態は、ストリーム自身の **mbstate_t** オブジェクトによって記述されます。バイト入力関数は、 **fgetc** 関数の連続呼び出しによって読み込みが実行された場合と同様に、ストリームから文字を読み込みます。

ワイド文字出力関数はワイド文字をマルチバイト文字に変換し、**fputwc** 関数の連続呼び出しによって書き込まれた場合と同様に、ストリームに書き込みます。個々の変換は **wcrtomb** 関数の呼び出しによる変換と同様に行われます。変換状態は、ストリーム自身の **mbstate_t** オブジェクトによって記述されます。バイト出力関数は、**fputc** 関数の連続呼び出しによって書き込まれた場合と同様に、ストリームに文字を書き込みます。

一部のバイト入出力関数では、マルチバイト文字とワイド文字間の変換も実行する場合があります。これらの変換も、 **mbrtowc** 関数および **wcrtomb** 関数の呼び出しによって変換された場合と同様に行われます。

基礎になる **mbrtowc** 関数から提供される文字シーケンスが有効な (汎用) マルチバイト文字を形成しない場合、または、基礎になる **wcrtomb** 関数に渡されるコード値が有効な (汎用) マルチバイト文字に対応しない場合に、エンコーディングエラーが発生します。ワイド文字入出力関数およびバイト入出力関数は、エンコーディングエラーが発生した場合に限り、**errno** 内にマクロ **EILSEQ** の値を格納します。

環境の制限

FOPEN_MAX の値は、3 つの標準テキストストリームを含み、8 以上です。

関数

ヘッダーファイル **stdio.h** には次の関数が定義されています。

関数	説明
clearerr	ストリームの EOF インジケータとエラーインジケータをクリアします。
fclose	ストリームをフラッシュし、関連付けられたファイルを閉じます。
feof	ストリームのファイルの終わりをテストします。
ferror	ストリームのエラーインジケータをテストします。
fflush	ファイルをフラッシュします。
fgetc	入力ストリームの次の文字を読み込みます。
fgetpos	ファイル位置インジケータの値を格納します。
fgets	行末が復帰文字である入力ストリーム から指定された文字数**-1** までの文字数の文字を読み込みます。
fopen	ファイルを開きます。
fprintf	特定のストリームに出力を行います。
fputc	出力ストリームに文字を書き込みます。
fputs	出力ストリームに文字列を書き込みます。
fread	ストリームから配列に読み込みます。
freopen	ファイルを開きます。
fscanf	特定のストリームから入力を読み込みます。
fseek	文字列のファイル位置インジケータを設定します。
fsetpos	ストリームに **mbstate_t** オブジェクトとファイル位置インジケータを設定します。
ftell	ストリームのファイル位置インジケータの現在の値を取得します。
fwrite	配列からストリームに書き込みます。
getc	入力ストリームから文字を読み込みます。
getchar	標準入力ストリームから次の文字を読み込みます。
gets	入力ストリームから文字を配列に読み込みます。
perror	**errno** 内のエラー番号をエラーメッセージにマップします。
printf	標準出力に出力を行います。
putc	出力ストリームに文字を書き込みます。
putchar	**stdout** に文字を書き込みます。
puts	文字列を出力します。
remove	ファイルを削除します。
rename	ファイル名を変更します。
rewind	ストリームの先頭にファイル位置インジケータを設定します。
scanf	標準入力から入力を読み込みます。
setbuf	ストリームのバッファを設定します。
setvbuf	ストリームのバッファモードとサイズを設定します。
snprintf	指定の配列に出力を行います。
sprintf	指定の配列に出力を行います。
sscanf	文字列から入力を読み込みます。

tmpfile	一時バイナリファイルを作成します。
tmpnam	一時ファイル名を作成します。
ungetc	入力ストリームに文字を押し戻します。
vfprintf	指定のストリームに出力を行います。
vprintf	指定のストリームに出力を行います。
vsnprintf	指定の配列に出力を行います。
vsprintf	指定の配列に出力を行います。

マクロ

ヘッダーファイル **stdio.h** には次のマクロが定義されています。

マクロ	説明
_IOFBF	**setvbuf** の第 3 引数に適している整数の定数式に展開されます。
_IOLBF	**setvbuf** の第 3 引数に適している整数の定数式に展開されます。
_IONBF	**setvbuf** の第 3 引数に適している整数の定数式に展開されます。
BUFSIZ	**setbuf** 関数で使用するバッファのサイズである整数の定数式に展開されます。
EOF	**int** で負の値を持つ整数の定数式に展開されます。これは、ファイルの終わりを示すために複数の関数によって返されます。
FILENAME_MAX	整数の定数式に展開され、実装で保証されている開くことが可能な最長のファイル名文字列を格納するために必要な配列 **char** のサイズを定義します。
FOPEN_MAX	実装で保証されている同時に開くことが可能なファイルの最大の個数である整数の定数式に展開されます。
L_tmpnam	**tmpnam** 関数で生成する一時ファイル名文字列を保持するのに十分な大きさを示す整数の定数式に展開されます。
SEEK_CUR	**fseek** での使用に適した異なる値を持つ整数の定数式に展開されます。
SEEK_END	**fseek** での使用に適した異なる値を持つ整数の定数式に展開されます。
SEEK_SET	**fseek** での使用に適した異なる値を持つ整数の定数式に展開されます。
TMP_MAX	**tmpnam** 関数で生成可能なファイル名の最大の個数である整数の定数式に展開されます。
stderr	標準エラーに関連付けられているファイルオブジェクトを指します。
stdin	標準入力に関連付けられているファイルオブジェクトを指します。
stdout	標準出力に関連付けられているファイルオブジェクトを指します。

宣言される型

ヘッダーファイル **stdio.h** には次の型が定義されています。

型	説明
size_t	符号なし int - **sizeof** 演算子の結果です。
FILE	オブジェクト - ストリームの制御に必要なすべての情報を記録します。
fpos_t	オブジェクト - ファイル内のすべての位置を一意に指定するのに必要なすべての情報を記録します。

移植性

関数 **vsscanf**()、**vfscanf**()、**vscanf**() は、Ch ではサポートされません。

clearerr

構文

#include <**stdio.h**>
void clearerr(**FILE** **stream*);

Purpose
目的

ストリームの EOF インジケータとエラーインジケータをクリアします。

戻り値

clearerr 関数は値を返しません。

パラメータ

stream ストリームへのポインタ。

説明

clearerr 関数は *stream* が指すストリームの EOF インジケータとエラーインジケータをクリアします。

例

```
/* clearerr() resets  the error indicator and  EOF
   to 0 on the same stream */
#include <stdio.h>

int main() {
    FILE *fp;
    char ch;

    if((fp = fopen("clearerr.c","r")) == NULL) {
        fprintf(stderr, "Error: cannot open file clearerr.c for reading\n");
        exit(1);
    }
    while(!feof(fp)) {
        ch = fgetc(fp);
    }
    printf("feof(fp) is %2d before clearerr(fp) is called.\n", feof(fp));
    clearerr(fp);
    printf("feof(fp) is %2d after clearerr(fp) is called.\n", feof(fp));
 }
```

出力

```
feof(fp) is 16 before clearerr(fp) is called.
feof(fp) is  0 after clearerr(fp) is called.
```

関連項目

fclose

構文

#include <**stdio.h**>
int fclose(**FILE** **stream*);

目的

ストリームをフラッシュし、関連付けられているファイルを閉じます。

戻り値

ストリームが正常に閉じた場合、関数 **fclose** はゼロを返します。エラーが検出された場合は、 **EOF** が返されます。

パラメータ

stream ストリームへのポインタ。

説明

関数 **fclose** は、引数 *stream* が指すストリームをフラッシュし、ストリームに関連付けられたファイルを閉じます。ストリームに書き込まれてないバッファ内のデータは、ホスト環境へ配信されてファイルに書き込まれ、読み込まれていないバッファ内のデータは破棄されます。ストリームとファイルの関連付けが解除されます。関連付けられたバッファが自動的に割り当てられていた場合は、その割り当ても解除されます。

例

clearerr() を参照してください。

出力

関連項目

feof

構文
#include <**stdio.h**>
int feof(**FILE** **stream*);

目的
ストリームのファイルの終わりをテストします。

戻り値
feof 関数は、*stream* に対して EOF インジケータが設定されている場合に限り、ゼロ以外を返します。

パラメータ
stream ストリームへのポインタ。

説明
feof 関数は、*stream* が指すストリームの EOF インジケータをテストします。

例
clearerr() を参照してください。
出力
関連項目

ferror

構文

#include <**stdio.h**>
int ferror(**FILE** **stream*);

目的

ストリームのエラーインジケータをテストします。

戻り値

ferror 関数は、*stream* に対してエラーインジケータが設定されている場合に限り、ゼロ以外を返します。

パラメータ

stream ストリームへのポインタ。

説明

ferror 関数は *stream* が指すストリームのエラーインジケータをテストします。

例

clearerr() を参照してください。

出力

関連項目

fflush

構文

#include <**stdio.h**>
int fflush(**FILE** **stream*);

目的

ファイルをフラッシュします。

戻り値

fflush 関数は、ストリームのエラーインジケータを設定し、書き込みエラーが発生した場合は **EOF** を返し、その他の場合はゼロを返します。

パラメータ

stream ストリームへのポインタ。

説明

fflush 関数は、*stream* が出力ストリーム、または最新の操作で入力のなかった更新ストリームを指している場合、そのストリームに書き込まれていないデータをホスト環境に配信してファイルに書き込みます。その他の場合、動作は不定です。

stream が null ポインタである場合、 **fflush** 関数は、上記に定義された動作の対象となるすべてのストリームにこのフラッシュ動作を実行します。

例

fwrite() を参照してください。

出力

関連項目

fgetc

構文

#include <**stdio.h**>
int fgetc(FILE **stream*);

目的

ストリームから入力を読み込みます。

戻り値

EOF インジケータが設定されている場合、またはそのストリームがファイルの終わりに位置する場合、ストリームの EOF インジケータが設定され、 **fgetc** は **EOF** を返します。その他の場合、 **fgetc** 関数は *stream* が指す入力ストリームの次の文字を返します。読み込みエラーが発生した場合は、ストリームにエラーインジケータが設定され、 **fgetc** は **EOF** を返します。

パラメータ

stream ストリームへのポインタ。

説明

stream が指す入力ストリームに EOF インジケータが設定されておらず、次の文字が存在する場合、 **fgetc** 関数は **int** に変換された **unsigned char** として該当の文字を取得し、そのストリームに関連付けられているファイル位置インジケータ (定義されている場合) を先へ進めます。

例

```
/* a sample program that reads a file then
displays it's content. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *stream;
    char ch;

    /* open file for reading only */
    if((stream=fopen("fgetc.c","r")) == NULL){
        printf("cannot open file\n");
        exit(1);
    }
    ch = fgetc(stream);
    printf("%c\n",ch);
    fclose(stream);
}
```

出力

```
/
```

関連項目

fgetpos

構文

#include <**stdio.h**>

int fgetpos(FILE * *restrict stream*, **fpos_t** * *resitrict pos*);

目的

ファイル位置インジケータの現在の値を格納します。

戻り値

成功すると、 **fgetpos** 関数は 0 を返します。失敗すると、 **fgetpos** 関数はゼロ以外を返し、**errno** 内に実装で定義される正の値を格納します。

パラメータ

stream ストリームへのポインタ。

pos ファイル位置インジケータの現在の値を格納するオブジェクトへのポインタ。

説明

fgetpos 関数は、*stream* が指すストリームの解析状態 (存在する場合) とファイル位置インジケータの現在の位置を、*pos* が指すオブジェクトに格納します。格納される値には、 **fsetpos** 関数がストリームの位置を **fgetpos** 関数の呼び出し時の位置に変更する際に使用できる、未指定の情報を含みます。

例

```
/* a sample program that uses the fgetpos() to read the fourth
   character from a file. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *stream;
    fpos_t pos;
    char ch;
    long int offset = 3;
    /* open file for reading */
    if((stream = fopen("fgetpos.c","r")) == NULL){
        printf("cannot open file\n");
        exit(1);
    }
    fseek(stream, offset ,SEEK_SET);
    fgetpos(stream,&pos);
    fsetpos(stream,&pos);
    fread(&ch,sizeof(char),1,stream);   //read a character
    printf("The current character in file is '%c'\n",ch);
    fclose(stream);
}
```

出力

```
The current character in file is 'a'
```

関連項目

fgets

構文

#include <**stdio.h**>
char ***fgets**(**char** * *restrict s*, **int** *n*, **FILE** **restrict stream*);

目的

ストリームからの入力をメモリに読み込みます。

戻り値

成功すると、 **fgets** 関数は *s* を返します。ファイルの終わりに到達し、配列に読み込まれた文字がなかった場合、配列の内容は変更されずに null ポインタが返されます。操作中に読み込みエラーが発生した場合、配列の内容は不定となり null ポインタが返されます。

パラメータ

s 読み込んだ文字列の格納先配列へのポインタ。

n 読み込み可能な最大文字数-1 を表す整数へのポインタ。

stream ストリームへのポインタ。

説明

fgets 関数は、最大で *n* が指定するよりも 1 文字少ない文字数を、*stream* が指すストリームから *s* が指す配列へ読み込みます。改行文字 (保持されている場合) またはファイルの終わりの後は、追加文字は読み込まれません。null 文字は、配列に最後の文字が読み込まれた直後に書き込まれます。

例

出力

関連項目

fopen

構文

#include <**stdio.h**>
FILE *fopen(**const char** **filename*, **const char** **mode*);

目的

ファイルを開きます。

戻り値

fopen 関数はストリームを制御するオブジェクトへのポインタを返します。開く操作に失敗すると、**fopen** は null ポインタを返します。

パラメータ

filename ファイル名を表す文字列へのポインタ。

mode オープンモード (説明参照) を表す文字列へのポインタ。

説明

fopen 関数は名前が *filename* が指す文字列であるファイルを開き、ストリームをそのファイルに関連付けます。

引数 *mode* は文字列を指します。文字列が次のいずれかである場合、ファイルは指定されたモードで開かれます。それ以外の場合、動作は不定です。

r	テキストファイルを読み込み用に開きます。
w	テキストファイルを書き込み用に長さゼロに切り捨てるか、または作成します。
a	追加。テキストファイルをファイルの終わりへの書き込み用に開くか、または作成します。
rb	バイナリファイルを読み込み用に開きます。
wb	バイナリファイルを書き込み用に長さゼロに切り捨てるか、または作成します。
ab	追加。バイナリファイルをファイルの終わりへの書き込み用に開くか、または作成します。
r+	テキストファイルを更新 (読み込み/書き込み) 用に開きます。
w+	テキストファイルを更新用に長さゼロに切り捨てるか、または作成します。
a+	追加。テキストファイルを更新 (ファイルの終わりに書き込み) 用に開くか、または作成します。
r+b または **rb+**	バイナリファイルをて更新 (読み込み/書き込み) 用に開きます。

w+b または **wb+**	バイナリファイルを更新用に長さゼロに切り捨てるか、または作成します。
a+b または **ab+**	追加。バイナリファイルを更新 (ファイルの終わりに書き込み) 用に開くか、または作成します。

ファイルを読み込みモード (引数 mode の最初の文字が**'r'**) で開いたときにファイルが存在しないか読み込みできない場合、操作は失敗します。

追加モード (引数 *mode* の最初の文字が (**'a'**) でファイルを開くと、以降のファイルへのすべての書き込みは、**fseek** 関数の呼び出しが介在するかどうかにかかわらず、強制的に現在のファイルの終わりに書き込まれます。一部の実装では、バイナリファイルを書き込みモード (引数 *mode* の値を示す上記のリストで 2 番目または 3 番目の文字が**'b'**) で開くと、null 文字の埋め込みにより、ストリームのファイル位置インジケータの初期位置が、最後に書かれたデータより後に置かれる場合があります。ファイルを更新モード (上記のリストにある引数 *mode* の値の 2 番目または 3 番目の文字が**'+'**) で開くと、入力、出力のどちらの操作も関連付けられたストリーム上で実行されます。ただし、関数 **fflush** またはファイル位置決め関数 (**fseek** 、 **fsetpos** 、または **rewind**) への呼び出しが介在しない場合、出力直後に入力操作を行ってはなりません。また、入力操作がファイルの終わりに達した場合を除き、ファイル位置決め関数の呼び出しが介在しない場合、入力直後に出力操作を行ってはなりません。

代わりに、実装方法によっては、テキストファイルを更新モードで開く (または作成する) ことで、バイナリストリームを開く (または作成する) ことができます。ファイルが開かれると、対話型デバイスを参照しないことが確認できた場合に限り、ストリームはフルバッファリングされます。ストリームのエラーインジケータと EOF インジケータはクリアされます。

例

clearerr() を参照してください。

出力

関連項目

fprintf

構文

#include <**stdio.h**>
int fprintf(FILE **restrict stream,* **const char** **restrict format,* ...);

目的

特定の形式を使用してストリームに出力を書き込みます。

戻り値

fprintf 関数は、送信された文字数を返すか、出力エラーまたはエンコーディングエラーが発生した場合は負の値を返します。

パラメータ

stream ストリームへのポインタ。

format 書式文字列へのポインタ。

説明

fprintf 関数は、後続の引数を出力に変換する方法を指定する *format* が指す文字列に従って、*stream* が指すストリームに出力を書き込みます。書式に対して引数が不足している場合、動作は不定です。引数があるにもかかわらず書式が終ってしまった場合、余分な引数は評価 (通常どおり) されますが無視されます。**fprintf** 関数は、書式設定文字列の終わりに達すると戻ります。

書式は、初期シフト状態の開始と終了を持つ、マルチバイト文字のシーケンスでなければなりません。書式は、ゼロ個以上の次のディレクティブで構成されます。通常のマルチバイト文字 (%を除く) は出力ストリームにそのままコピーされます。変換指定はそれぞれ後続のゼロ個以上の引数を取得し、適切な場合は対応する変換指定子に従って変換し、結果を出力ストリームに書き込みます。

各変換指定は、文字%で開始されます。%の後に、以下の情報を順番に指定します。

– 変換指定の意味を変更する 0 個以上の *flags*(順不同)。

– (オプション) 最小フィールド幅。変換後の値に含まれる文字数がフィールド幅に満たない場合、既定では、フィールド幅まで左側に (または後述する左揃えフラグが指定されている場合は右側に) 空白が埋め込まれます。フィールド幅はアスタリスク (後述) または 10 進整数の形をとります。

– (オプション) **d**、**i**、**o**、**u**、**x**、および **X** 変換で表示される最小桁数、**a**、**A**、**e**、**E**、**f**、および **F** の変換で小数点以下に表示される桁数、**g** および **G** 変換での最大有効桁数、または **s** 変換で文字列から書き込まれる最大文字数などの数値を指定する精度。精度は、アスタリスク (後述) またはオプションの 10 進整数のいずれかの前にピリオド (.) を置く形で表されます。ピリオド

しか指定されていない場合、精度はゼロとして扱われます。上記以外の変換指定子で精度を指定した場合、動作は不定です。

– (オプション) 引数のサイズを指定する長さ修飾子。

– 適用する変換の種類を指定する変換指定子文字。

上記の説明にあるように、フィールド幅または精度あるいはその両方をアスタリスクで示すことができます。この場合、フィールド幅または精度は **int** 型の引数で指定します。フィールド幅または精度あるいはその両方を指定する引数は、変換される引数 (もしあれば) の前に (この順番で) 置く必要があります。フィールド幅の引数を負の値で指定すると、正のフィールド幅の前に"-" (マイナス) フラグが付けられます。精度の引数が負の場合、精度が省略されていると見なされます。

各フラグ文字とその意味は次のとおりです。

- 変換結果をフィールド内に左揃えで表示します (このフラグを指定しない場合は右揃えになります)。

+ 符号付き変換の結果は、先頭が常に "+" 符号または "+" 符号になります (このフラグを指定しない場合、負の値が変換される場合のみ先頭が符号になります)。

space 符号付き変換の最初の文字が符号でない場合、または、符号付き変換の結果に文字がない場合、結果の前に空白が付加されます。*space* フラグと+フラグの両方を指定すると、*space* フラグが無視されます。

結果は "代替形式" に変換されます。**o** 変換では、必要な場合にのみ精度を上げ、結果の最初の桁を強制的に 0 にします (値と精度の両方が 0 の場合は、0 が 1 つ出力されます)。**x** (または **X**) 変換では、結果がゼロ以外の場合、前に **0x** (または **0X**) が付加されます。**a**、**A**、**e**、**E**、**f**、**F**、**g**、および **G** 変換では、浮動小数点数の変換結果には、小数点以下の数字がない場合でも常に小数点文字が表示されます (通常、変換結果に小数点が表示されるのは、小数点以下の数字がある場合のみです)。**g** および **G** の変換では、末尾のゼロは結果からは削除されません。その他の変換では、このフラグの動作は不定です。

0 **d**、**i**、**o**、**u**、**x**、**X**、**a**、**A**、**e**、**E**、**f**、**F**、**g**、および **G** 変換の場合、フィールド幅を埋めるために空白を埋め込むのではなくゼロを先頭に (符号または基数が指定されている場合はその後に続けて) 付けます。ただし、無限大または NaN を変換する場合を除きます。**0** フラグと-フラグの両方を指定した場合、**0** フラグが無視されます。**d**、**i**、**o**、**u**、**x**、および **X** 変換では、精度を指定した場合、**0** フラグが無視されます。その他の変換では、このフラグの動作は不定です。

長さ修飾子とそれぞれの意味は以下のとおりです。

hh 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、または **X** の場合は、その指定子が **signed char** 型または **unsigned char** 型の引数に適用されることを指定します (引数は整数の上位変換に応じて上位変換されますが、値は **signed char** 型または **unsigned char** 型に上位変換してから出力する必要があります)。または、**n** の場合は、それが **signed char** 型の引数へのポインタに適用されることを指定します。

h 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、または **X** の場合は、その指定子が **short int** 型または **unsigned short int** 型の引数に適用されることを指定します (引数は整数の上位変換に応じて上位変換されますが、値は **short int** 型または **unsigned short int** 型に上位変換してから出力する必要があります)。または、**n** の場合は、それが **short int** 型の引数へのポインタに適用されることを指定します。

l (エル) 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、または **X** の場合は、その指定子が **long int** 型または **unsigned long int** 型の引数に適用されることを指定します。**n** の場合は **long int** 型の引数へのポインタに、**c** の場合は **wint_t** 型の引数に、**s** の場合は **wchar_t** 型の引数へのポインタにそれぞれ適用されることを指定します。または、後続の **a**、**A**、**e**、**E**、**f**、**F**、**g**、または **G** 変換指定子に影響を与えないことを指定します。

ll (エルエル) 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、または **X** の場合は、その指定子が **long long int** 型または **unsigned long long int** 型の引数に適用されることを指定します。または、**n** の場合は、それが **long long int** 型の引数へのポインタに適用されることを指定します。

j 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、または **X** の場合は、その指定子が **intmax_t** 型または **uintmax_t** 型の引数に適用されることを指定します。または、**n** の場合は、**intmax_t** 型の引数へのポインタに適用されることを指定します。

z 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、または **X** の場合は、その指定子が **size_t** 型の引数または対応する符号付き整数型引数に適用されることを指定します。または、n の場合は、それが **size_t** 型の引数に対応する符号付き整数型引数に適用されることを指定します。

t 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、または **X** の場合は、その指定子が **ptrdiff_t** 型の引数または対応する符号なし整数型引数に適用されることを指定します。または、**n** の場合は、それが **ptrdiff_t** 型の引数へのポインタに適用されることを指定します。

L 後続の変換指定子が **a**、**A**、**e**、**E**、**f**、**F**、**g**、または **G** の場合は、その指定子が **long double** 型の引数に適用されることを指定します。

長さ修飾子を上記の変換指定子以外の変換指定子と一緒に指定した場合、動作は不定です。

変換指定子とそれぞれの意味は次のとおりです。

d,i **int** 型の引数を *[-]dddd* という形式の符号付き 10 進数に変換します。精度は表示する最小桁数を指定します。変換対象の値を少ない桁数で表現できる場合は、先頭にゼロが埋め込まれます。既定の精度は 1 です。精度 0 でゼロの値を変換した結果は、文字数ゼロです。

o,u,x,X **unsigned int** 型の引数を *dddd* という形式の符号なし 8 進数 (**o**)、符号なし 10 進数 (**u**)、または符号なし 16 進数 (**x** または **X**)) の表記に変換します。**x** 変換には **abcdef** の文字が、**X** 変換には **ABCDEF** の文字が使用されます。精度は表示する最小桁数を指定します。変換対象の値を少ない桁数で表現できる場合は、先頭にゼロが埋め込まれます。既定の精度は 1 です。精度 0 でゼロの値を変換した結果は、文字数ゼロです。

f,F 浮動小数点数を表す **double** 型引数を *[-]ddd.ddd* という形式の 10 進数表記に変換します。小数点以下の桁数は精度の指定に等しくなります。精度を指定しない場合、精度は 6 と見なされます。精度を 0 に指定して#フラグを指定しない場合、小数点文字は表示されません。小数点文字を表示する場合は、小数点の前に少なくとも 1 桁が表示されます。値は、適切な桁数に丸められます。

無限大を表す **double** 型引数は、*[-]***inf** または *[-]***infinity** のいずれかの形式で変換されます。どちらの形式になるかは実装で定義されます。NaN を表す **double** 型引数は、*[-]***nan** または *[-]***nan***(n-char-sequence)* のいずれかの形式で変換されます。どちらの形式になるか、および n-char シーケンスの意味は、実装で定義されます。**F** 変換指定子は、**inf**、**infinity**、**nan** の代わりに、それぞれ **INF**、**INFINITY**、**NAN** を生成します。

e,E 浮動小数点数を表す double 型引数を *[-]d.ddd***e**±*dd* という形式に変換します。小数点文字の前に 1 桁 (引数がゼロ以外の場合はゼロ以外) が表され、小数点以下の桁数は精度の指定に等しくなります。精度を指定しない場合は、6 として扱われます。精度 0 を指定し、#フラグを指定しない場合、小数点文字は表示されません。値は、適切な桁数に丸められます。**E** 変換指定子は、指数を示す **e** の代わりに、**E** の付いた数字を出力します。指数には常に 2 桁以上が含まれ、指数を表すのに必要な分だけ桁が増やされます。値がゼロなら、指数もゼロです。
無限大または NaN を表す **double** 型引数は、**f** または **F** の変換指定子の形式に変換されます。

g,G 浮動小数点数を表す **double** 型引数を、有効桁数を示す精度を付けて、**f** または **e**(または **G** 変換指定子の場合は、**F** または **E**) という形式に変換します。精度 0 の場合は、1 として扱われます。使用される形式は、変換される値によって決まります。変換結果の指数が-4 よりも小さいか、精度に等しいまたはそれよりも大きい場合にのみ、**e** (または **E**) が使用されます。末尾のゼロは、#フラグが指定されていない限り変換結果の小数部から削除されます。小数点文字が表示されるのは、小数点以下の桁が存在する場合のみです。
無限大または NaN を表す **double** 型引数は、**f** または **F** の変換指定子の形式に変換されます。

a,A 浮動小数点数を表す **double** 型引数を、*[-]***0x***h.hhhh***p**±*d* という形式に変換します。小数点文字の前に 16 進数が 1 桁 (引数が正規化された浮動小数点数の場合はゼロ以外の値になり、それ以外の場合は未指定) が表され、小数点以下の桁数は精度の指定に等しくなります。精度が指定されておらず、**FLT_RADIX** が 2 の累乗である場合、精度は値を正確に表現できます。精度が指定されておらず、**FLT_RADIX** が 2 の累乗でない場合、精度は **double** 型の値を判別します。ただし、末尾のゼロは削除されます。精度に 0 を指定し、#フラグを指定しない場合、小数点は表示されません。a 変換には **abcdef** の文字が、**A** 変換には **ABCDEF** の文字が使用されます。A 変換指定子は、**x** および **p** ではなく、**X** および **P** を付けた数字を生成します。指数には常に 1 桁以上が含まれ、2 の 10 進指数を表現するのに必要な分だけ桁数が増やされます。値がゼロなら、指数もゼロです。
無限大または NaN を表す **double** 型引数は、**f** または **F** の変換指定子の形式に変換されます。

c 長さ修飾子 **l** を指定しない場合、int 型引数は **unsigned char** 型に変換され、その結果の文字が書き込まれます。
長さ修飾子 **l** を指定した場合、**wint_t** 引数は、精度の指定を付けずに、**wchar_t** の 2 つの要素配列 (最初の要素には lc 変換指定への **wint_t** 引数が含まれ、2 番目の要素には null ワイド文字が含まれる) の最初の要素を参照する引数を持つ **lc** 変換指定で変換された場合と同様に変更されます。

s 長さ修飾子 **l** を指定しない場合、引数は文字型配列の最初の要素へのポインタでなければなりません。配列からの文字は、終了 null 文字まで (null 文字は含まない) 書き出されます。精度を指定した場合は、指定以上の文字は書き込まれません。精度を指定しない、または配列サイズより大きい精度を指定した場合は、null 文字まで配列に含める必要があります。
長さ修飾子 **l** を指定した場合、引数は **wchar_t** 型配列の最初の要素へのポインタでなければなりません。この配列からのワイド文字は、終端の null ワイド文字 (その null 文字を含む) までマルチバイト文字へ変換されます。各文字は **wcrtomb** への関数呼び出しで変換された場合と同様に変換され、**mbstate_t** オブジェクトで表される変換状態は、最初のワイド文字が変換される前にゼロに初期化されます。変換結果のマルチバイト文字は、終端の null 文字 (バイト) まで書き込まれます (その終端文字は含まない)。精度を指定しない場合、配列に null ワイド文字が含まれている必要があります。精度を指定した場合、指定した以上の文字数 (バイト数) は書き出されません (シフトシーケンスが含まれている場合は、シフトシーケンスが含まれます)。また、関数が配列の終わりの 1 つ後にあるワイド文字にアクセスする必要がある場合、配列には、精度で指定されているマルチバイト文字のシーケンスと等しい長さになるように、null ワイド文字が 1 つ含まれている必要があります。マルチバイト文字が部分的に書き込まれることはありません。

p 引数は **void** へのポインタでなければなりません。ポインタの値は、実装によって定義された方法で、出力文字シーケンスに変換されます。

n この引数は符号付き整数へのポインタでなければなりません。その整数に書き込まれるのは、**fprintf** へのこの呼び出しでこれまでに出力ストリームに書き込まれた文字の数です。引数は変換されませんが、使用されます。変換指定になんらかのフラグ、フィールド幅、または精度が含まれている場合、動作は不定です。

% %文字を書き込みます。引数は変換されません。完全な形で変換指定する場合は、%%と指定する必要があります。

変換指定が正しくない場合、動作は不定です。引数の型と対応する変換指定が一致しない場合、動作は不定です。

フィールド幅を指定しない場合、または指定したフィールド幅が小さい場合でも、フィールドが切り詰められることはありません。変換結果がフィールド幅より大きくなった場合は、変換結果が表示されるようにフィールド幅が広げられます。

a 変換および **A** 変換では、**FLT_RADIX** が 2 の累乗である場合、値は精度に指定した 16 進浮動小数点数に正しく丸められます。

FLT_RADIX が 2 の累乗でない場合、結果は 2 つの隣接する数字のいずれかになり、精度に指定した 16 進浮動小数点形式で表されます。ただし、誤差を表す符号は、現在の丸め方向に一致します。

e、**E**、**f**、**F**、**g**、および **G** 変換では、10 進有効桁数が最大 **DECIMAL_DIG** である場合、結果は正しく丸められます。10 進有効桁数が **DECIMAL_DIG** を超え、ソースの値が **DECIMAL_DIG** の桁数で正確に表現できる場合、結果は末尾にゼロを含む形で正確に表現されます。そうでない場合、ソース

の値は L < U の関係を持つ隣接する 2 つの 10 進数文字列 (どちら **DECIMAL_DIG** 桁の有効桁数を持つ) で丸められます。その結果の 10 進数文字列 D の値は、L ≤ D ≤ U を満たします。ただし、誤差を表す符号は、現在の丸め方向に一致します。

例
出力
関連項目

fputc

構文

#include <**stdio.h**>
int ***fputc**(**int** *c*, **FILE** **stream*);

目的

stdout に出力を行います。

戻り値

fputc 関数は書き込まれた文字を返します。書き込みエラーが発生した場合は、ストリームのエラーインジケータが設定され、 **fputc** が **EOF** を返します。

パラメータ

c 出力する文字。

stream ストリームへのポインタ。

説明

fputc 関数は、*stream* が指す出力ストリーム内の、そのストリームに関連付けられているファイル位置インジケータ (定義されている場合) が指定する位置に、*c* に指定された文字 (**unsigned char** に変換される) を書き込み、適切にインジケータを進めます。ファイルが位置決め要求をサポートできない場合、またはストリームを書き込みモードで開いた場合、文字は出力ストリームに追加されます。

例

```
/* a sample program that opens a temporary file and
write a character and a string into it. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *stream;
    char ch='A';
    char *name;

    if((name = tmpnam(NULL)) == NULL)
        printf("Error in creating temp file.\n");
    /* open file for writing only */
    if((stream=fopen(name,"w")) == NULL) {
      printf("cannot open file\n");
      exit(1);
    }
    else {
        fputc(ch,stream);
        fputs("This is a test\n",stream);
        fputs("This is a test\n",stdout);
        fclose(stream);
```

```
    }
}
```

出力

```
This is a test
```

関連項目

fputs

構文

#include <**stdio.h**>
int ***fputs**(**const char** * *restrict s*, **FILE** * *restrict stream*);

目的

文字列を出力します。

戻り値

fputs 関数は *s* が指す文字列を *stream* が指すストリームに書き込みます。終端の null 文字は書き込まれません。

パラメータ

s 出力する文字列へのポインタ。

stream ストリームへのポインタ。

説明

fputs 関数は、*s* が指す文字列を *stream* が指すストリームに書き込みます。終端の null 文字は書き込まれません。

例

fputc() を参照してください。

出力

関連項目

fputc() を参照してください。

fread

構文

#include <**stdio.h**>
int fread(void * *restrict ptr*, **size_t** *size*, **size_t** *nmemb*, **FILE** * *restrict stream*);

目的

入力された文字列を配列に読み込みます。

戻り値

fread 関数は読み込みに成功した要素数を返します。この要素数は、読み込みエラーが発生したかファイルの終わりに到達した場合、*nmemb* 未満である可能性があります。*size* または *nmemb* がゼロである場合、*fread* はゼロを返し、配列の内容およびストリームの状態はそのまま維持されます。

パラメータ

ptr 読み込んだ文字列を格納する配列へのポインタ。

size バイト単位の要素のサイズ。

nmemb 読み出す最大要素数。

stream ストリームへのポインタ。

説明

fread 関数は、*size* で指定されたサイズの要素を最大数 *nmemb* まで、*stream* が指すストリームから *ptr* が指す配列に読み込みます。ストリームのファイル位置インジケータ (定義されている場合) は、読み込みに成功した文字数の分だけ進みます。エラーが発生した場合、ストリームのファイル位置インジケータの結果の値は不定です。一部の要素が読み込まれる場合、その値は不定です。

例

出力

関連項目

freopen

構文

#include <**stdio.h**>
FILE ***freopen**(**const char** **filename*, **const char** **mode*, **FILE** * *restrict stream*);

目的

ファイルを開きます。

戻り値

freopen 関数は、開く操作に失敗すると、null ポインタを返します。それ以外の場合は、*stream* の値を返します。

パラメータ

filename ファイル名を表す文字列へのポインタ。

mode オープンモード (**fopen** 関数の説明参照) を表す文字列へのポインタ。

stream ストリームを指します。

説明

freopen 関数は名前が *filename* が指す文字列であるファイルを開き、*stream* が指すストリームをそのファイルに関連付けます。引数 *mode* は **fopen** 関数内と同様に使用されます。
filename が null ポインタである場合、 **freopen** 関数は、そのストリームに現在関連付けられているファイル名が使用された場合と同様に、ストリームのモードを *mode* で指定されるモードに変更するよう試行します。どのような状況でどのモード変更が許可されるか (許可されるモード変更がある場合) は、実装で定義されます。

freopen 関数は最初に、指定されたストリームに関連付けられているファイルを閉じます。ファイルを閉じる操作に失敗してもそのエラーは無視されます。ストリームのエラーインジケータと EOF インジケータはクリアされます。

例

出力

関連項目

fscanf

構文

#include <**stdio.h**>
int fscanf(FILE **restrict stream,* **const char** **restrict format,* ...);

目的

指定された形式を使用してストリームの入力を読み込みます。

戻り値

fscanf 関数は、変換前に入力エラーが発生した場合、マクロ **EOF** の値を返します。それ以外の場合は、代入された入力項目の数を返します。以前にマッチングエラーが発生している場合、この数は指定された入力項目の数よりも少ない可能性があり、ゼロになることもあります。

パラメータ

stream ストリームへのポインタ。

format 書式文字列へのポインタ。

説明

fscanf 関数は、*format* が指す文字列に従って、*stream* が指すストリームから入力を読み込みます。format は、変換された入力を受け取るためのオブジェクトへのポインタとして後続の引数を使用し、許容される入力シーケンスとそのシーケンスを変換して割り当てる方法を指定します。書式に対して引数が不足している場合、動作は不定です。引数があるにもかかわらず書式が終ってしまった場合、余分な引数は評価 (通常どおり) されますが、無視されます。

入力書式は、初期シフト状態の開始と終了を持つ、マルチバイト文字のシーケンスです。入力書式は、1 つ以上の空白文字、通常のマルチバイト文字 (%でも空白文字でもない)、または変換指定など、ゼロ個以上のディレクティブで構成されます。各変換指定は、文字%で開始されます。
%の後に、以下の情報を順番に指定します。

– (オプション) 代入抑止文字 “*****”。

– (オプション) フィールドの最大幅 (文字数) を指定するゼロ以外の小数整数。

– (オプション) 受信オブジェクトのサイズを指定する長さ修飾子。

– 適用する変換の種類を指定する変換指定子の文字。

fscanf 関数は入力形式の各ディレクティブを順番に実行します。この後で詳しく説明するように、ディレクティブにエラーがある場合、関数は終了します。エラーは、入力エラー (コーディングエラーの発生または使用できない入力文字が原因)、またはマッチングエラー (不適切な入力が原因) です。

空白文字で構成されるディレクティブは、読み込まれないで残っている最初の空白以外の文字まで、または読み込む文字がなくなるまで、入力を読み込んで実行されます。

通常のマルチバイト文字で構成されるディレクティブは、ストリームの次の文字を読み込むことによって実行されます。これらの文字のいずれかがディレクティブを構成する文字と異なる場合、ディレクティブの実行は失敗し、異なる文字以降の文字は読み込まれずに残ります。

変換指定のディレクティブは、以下の各指定子の説明にあるように、一連のマッチング入力シーケンスを定義します。変換指定は次のステップで実行されます。

[、**c**、**n** などの変換指定子が変換指定に含まれている場合を除き、入力した空白文字 (**isspace** 関数で指定される) はスキップされます。

n 変換指定子が変換指定に含まれている場合を除き、入力項目はストリームから読み込まれます。入力項目は、入力された最も長い文字シーケンス (指定したフィールド幅を超えず、マッチング入力シーケンスまたはその接頭語である) として定義されます。入力項目の後の最初の文字 (もしあれば) は読み込まれないで残ります。入力項目の長さが 0 の場合、ディレクティブの実行は失敗します。この状態はマッチングエラーです。ただし、ファイルの終わり (EOF)、エンコーディングエラー、またはストリームからの読み込みエラー (この場合は入力エラー) の場合を除きます。

変換指定子が%の場合を除き、入力項目 (%**n** ディレクティブの場合は、入力文字数) は、変換指定子に適した型に変換されます。入力項目がマッチングシーケンスではない場合、ディレクティブの実行は失敗します。この状態はマッチングエラーです。代入抑止が “*****” で表された場合を除き、変換結果は、引数 *format* の後で最初に出現する、まだ変換結果を受け取っていない引数が参照するオブジェクトに出力されます。このオブジェクトが適切な型でない場合、または変換結果をこのオブジェクトで表すことができない場合、動作は不定です。

長さ修飾子とそれぞれの意味は以下のとおりです。

hh 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、**X**、または **n** の場合、その指定子が **signed char** または **unsigned char** へのポインタ型を持つ引数に適用されることを指定します。

h 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、**X**、または **n** の場合、その指定子が **short int** または **unsigned short int** へのポインタ型を持つ引数に適用されることを指定します。

l (エル) 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、**X**、または **n** の場合、持つ引数に適用されることを指定します。**a**、**A**、**e**、**E**、**f**、**F**、**g**、または、**G** の場合は、**double** へのポインタ型を持つ引数に、**c**、**s**、または **[**の場合は、**wchar_t** へのポインタ型を持つ引数に適用されることを指定します。
その指定子が **long int** または **unsigned long** へのポインタ型を
a、**A**、**e**、**E**、**f**、**F**、**g**、または、**G** の場合は、
double へのポインタ型を持つ引数に、**c**、**s**、または **[**の場合は、
wchar_t へのポインタ型を持つ引数に適用されることを指定します。

ll (エルエル) 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、**X**、または **n** の場合、その指定子が
long long int または **unsigned long long int** へのポインタ型を持つ引数に適用されることを指定します。

j 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、**X**、または **n** の場合、
その指定子が **intmax_t** または **uintmax_t** へのポインタ型を持つ引数に適用されることを指定します。

z 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、**X**、または **n** の場合、
その指定子が **size_t** または対応する符号付き整数型へのポインタ型を持つ引数に適用されることを指定します。

t 後続の変換指定子が **d**、**i**、**o**、**u**、**x**、**X**、または **n** の場合、その指定子が **ptrdiff_t** または対応する符号なし整数型へのポインタ型を持つ引数に適用されることを指定します。

L 後続の変換指定子が **a**、**A**、**e**、**E**、**f**、**F**、**g**、または **G** の場合、その指定子が **long double** へのポインタ型を含む引数に適用されることを指定します。

長さ修飾子を上記の変換指定子以外の変換指定子と一緒に指定した場合、動作は不定です。

変換指定子とそれぞれの意味は次のとおりです。

d 10 進数の整数 (符号は任意) と一致します。入力形式は、引数 **base** に値 10 を指定した関数 **strtol** の変換対象のシーケンスと同じシーケンスである必要があります。対応する引数は、符号付き整数へのポインタ型でなければなりません。

i 整数 (符号は任意) と一致します。入力形式は、引数 **base** に値 0 を指定した関数 **strtol** の変換対象のシーケンスと同じシーケンスである必要があります。対応する引数は、符号付き整数へのポインタ型でなければなりません。

o 8 進数の整数 (符号は任意) と一致します。入力形式は、引数 **base** に値 8 を指定した関数 **strtoul** の変換対象のシーケンスと同じシーケンスである必要があります。対応する引数は、符号なし整数へのポインタ型でなければなりません。

u 10 進数の整数 (符号は任意) と一致します。入力形式は、引数 **base** に値 10 を指定した関数 **strtoul** の変換対象のシーケンスと同じシーケンスである必要があります。対応する引数は、符号なし整数へのポインタ型でなければなりません。

x 16 進数の整数 (符号は任意) と一致します。入力形式は、引数 **base** に値 16 を指定した関数 **strtoul** の変換対象のシーケンスと同じシーケンスである必要があります。対応する引数は、符号なし整数へのポインタ型でなければなりません。

a,e,f,g 浮動小数点数、無限、または NaN と一致します (符号は任意)。入力形式は、関数 **strtod** の変換対象のシーケンスと同じシーケンスである必要があります。対応する引数は、浮動少数点数へのポインタ型でなければなりません。

c フィールド幅に指定された数 (ディレクティブにフィールド幅の指定がない場合は 1) の文字シーケンスと一致します。

長さ修飾子 **l** を指定しない場合、対応する引数は、そのシーケンスを受け入れるだけの十分な大きさを持つ文字配列の先頭要素へのポインタでなければなりません。null 文字は追加されません。

l 長さ修飾子 l を指定した場合、入力は、初期シフト状態で始まるマルチバイト文字のシーケンスでなければなりません。シーケンス内の各マルチバイト文字は、**mbrtowc** への関数呼び出しを行った場合と同様にワイド文字へ変換され、変換状態は、最初のマルチバイト文字が変換される前にゼロに初期化された **mbstate_t** オブジェクトによって記述されます。対応する引数は、結果のワイド文字のシーケンスを受け入れるだけの十分な大きさを持つ **wchar_t** 配列の先頭要素へのポインタでなければなりません。null ワイド文字は追加されません。

s 空白文字でない文字のシーケンスに一致します。
長さ修飾子 **l** を指定しない場合、対応する引数は、シーケンスと終端 null 文字 (自動的に含まれる) を受け入れるだけの十分な大きさを持つ文字配列の先頭要素へのポインタでなければなりません。
長さ修飾子 **l** を指定した場合、入力は、初期シフト状態で始まるマルチバイト文字のシーケンスでなければなりません。各マルチバイト文字は、**mbrtowc** への関数呼び出しを行った場合と同じ様に、ワイド文字へ変換され、変換状態は、最初のマルチバイト文字が変換される前にゼロに初期化された **mbstate_t** オブジェクトによって記述されます。対応する引数は、シーケンスと終端 null 文字 (自動的に含まれる) を受け入れるだけの十分な大きさを持つ **wchar_t** 配列の先頭要素へのポインタでなければなりません。

[必要な文字セット (scanset) からの空でない文字シーケンスと一致します。
長さ修飾子 **l** を指定しない場合、対応する引数は、シーケンスと終端 null 文字 (自動的に含まれる) を受け入れるだけの十分な大きさを持つ文字配列の先頭要素へのポインタでなければなりません。
長さ修飾子 **l** を指定した場合、入力は、初期シフト状態で始まるマルチバイト文字のシーケンスでなければなりません。各マルチバイト文字は、**mbrtowc** への関数呼び出しを行った場合と同じ様に、ワイド文字へ変換され、変換状態は、最初のマルチバイト文字が変換される前にゼロに初期化された **mbstate_t** オブジェクトによって記述されます。対応する引数は、シーケンスと終端 null ワイド文字 (自動的に含まれる) を受け入れるだけの十分な大きさを持つ **wchar_t** 配列の先頭要素へのポインタでなければなりません。
変換指定子には、**format** 文字列内の後続のすべての文字を、対応する右角かっこ (**]**) まで含めます (角かっこも含む)。角かっこに囲まれた文字 (*scanlist*) は scanset を構成します。ただし、左角かっこの後にカレット (^) がある場合は、カレットと右角かっこに挟まれた scanlist に入っていないすべての文字が scanset を構成します。変換指定子が **[]** または **[^]** で始まる場合、右角かっこは scanlist 内に含まれ、この後に出現する右角かっこが、指定の終わりを示す対応する右角かっこになります。2 番目の右角かっこがない場合は、最初の右角かっこが、変換指定の終わりを示す右かっこと見なされます。scanlist に含まれる "-" の文字が 1 文字目ではなく、2 文字目 ("^" を 1 文字目として) でもなく、かつ末尾の文字でない場合、動作は実装で定義されます。

p 実装で定義されているシーケンスに一致します。これにより、関数 **fprintf** の **%p** 変換によって出力されるシーケンスと同じになります。対応する引数は、 **void** へのポインタのポインタでなければなりません。入力項目は、実装で定義された方法でポインタ値に変換されます。入力項目が、同じプログラムの実行中で以前に変換された値の場合、結果となるポインタはその値と等しくなければなりません。そうでない場合、**%p** 変換の動作は不定です。

n 入力は使用されません。対応する引数は、**fscanf** 関数のこの呼び出しでこれまでに入力ストリームから読み取った文字の数を示す、符号付き整数へのポインタでなければなりません。**%n** ディレクティブを実行しても、 **fscanf** 関数の実行完了時に返される代入回数は増えません。引数は変換されませんが、使用されます。変換指定に代入抑止文字またはフィールド幅が含まれている場合、動作は不定です。

% 単一の%文字と一致します。変換または代入は行われません。完全な形で変換指定する場合は、%%と指定する必要があります。

変換指定が正しくない場合、動作は不定です。

A、**E**、**F**、**G**、および **X** の変換指定子も有効です。それぞれ、**a**、**e**、**f**、**g**、および **x** と同じ動作をします。

入力中にファイルの終わりに達すると、変換は終了します。現在のディレクティブに一致する文字 (許可されている場合は先行の空白を除く) が読み込まれる前にファイルの終わりに達した場合、現在のディレクティブの実行は入力エラーで終了します。それ以外の場合、現在のディレクティブの実行がマッチングエラーで終了しなければ、以降のディレクティブ (**%n** を除く) の実行が入力エラーで終了します。

ディレクティブと一致しない場合、末尾の空白 (改行文字も含む) は読み込まれずに残ります。リテラルの一致と抑止された代入が成功したかどうかを直接確認するには、**%n** ディレクティブを使用する以外にありません。

入力文字が競合して変換が終了した場合、入力ストリーム内の競合する入力文字は読み込まれないままになります。

例
出力
関連項目

fseek

構文

#include <**stdio.h**>
int fseek(**FILE** **stream*, **long int** *offset*, **int** *whence*);

目的

ファイル位置インジケータの現在の値を格納します。

戻り値

関数 **fseek** は満たすことができない要求に対してのみゼロ以外の値を返します。

パラメータ

stream ストリームへのポインタ。

offset 基準点からのバイト数。

whence 基準点を表す整数。[説明] を参照してください。

説明

fseek 関数は *stream* が指すストリームのファイル位置インジケータを設定します。読み込みエラーまたは書き込みエラーが発生した場合は、ストリームのエラーインジケータが設定され、 **fseek** は失敗します。

バイナリストリームの場合、*whence* に指定した位置に *offset* 値を追加することにより、ファイルの先頭からの文字数で測定された新しい位置を取得します。*whence* が **SEEK_SET** の場合は、指定位置はファイルの先頭になります。**SEEK_CUR** の場合はファイル位置インジケータの現在の値、**SEEK_END** の場合はファイルの終わりになります。バイナリストリームは **SEEK_END** の *whence* 値による **fseek** の呼び出しを有効にサポートするとは限りません。

テキストストリームの場合、*offset* は、ゼロか、または同じファイルに関連付けられているストリームで以前に成功した **ftell** への関数呼び出しで返された値のどちらかにならなければなりません。*whence* は **SEEK_SET** でなければなりません。

新しい位置が決定した後、 **fseek** 関数の呼び出しに成功すると、ストリーム上の **ungetc** 関数の影響を元に戻し、ストリームの EOF インジケータをクリアして、新しい位置を設定します。 **fseek** の呼び出しに成功すると、更新ストリームの次の操作は、入力または出力のいずれかになります。

例

```
/* a sample program that reads the third charter
in a file. */
#include <stdio.h>
```

```
#include <stdlib.h>

int main() {
    FILE *stream;
    char ch;
    long int offset;

    offset = 2*sizeof(char);
    /* open file for reading and writing */
    if((stream = fopen("fseek.c","r")) == NULL){
      printf("cannot open file\n");
      exit(1);
    }
    fseek(stream,offset,0);     //search the third character
    fread(&ch,sizeof(char),1,stream);   //read a character
    printf("The third character in file is '%c'\n",ch);
    rewind(stream);                     //set file indicator to first
    fread(&ch,sizeof(char),1,stream);
    printf("The first character in file is '%c'\n",ch);
    fclose(stream);
}
```

出力

```
The third character in file is ' '
The first character in file is '/'
```

関連項目

fsetpos

構文

#include <**stdio.h**>
int fsetpos(**FILE** * *restrict stream*, **fpos_t** * *resitrict pos*);

目的

ファイル位置インジケータの現在の値を格納します。

戻り値

成功した場合、 **fsetpos** 関数は 0 を返します。失敗した場合、 **fsetpos** 関数は非ゼロを返し、実装で定義される正の値を **errno** に格納します。

パラメータ

stream ストリームへのポインタ。

pos ファイル位置インジケータの現在の値を格納するオブジェクトへのポインタ。

説明

fsetpos 関数は、*pos* が指すオブジェクトの値に従って、*stream* が指すストリームの **mbstate_t** オブジェクト (存在する場合) とファイル位置インジケータを設定します。これは、同じファイルに関連付けられたストリームで以前に成功した **fgetpos** 関数の呼び出しで取得された値でなければなりません。読み込みエラーまたは書き込みエラーが発生した場合は、ストリームのエラーインジケータが設定され、**fsetpos** は失敗します。

fsetpos 関数の呼び出しに成功すると、ストリーム上の **ungetc** 関数の影響を元に戻し、ストリームの EOF インジケータをクリアして、新しい解析状態と位置を設定します。 **fsetpos** の呼び出しに成功すると、更新ストリームの次の操作は、入力または出力のいずれかになります。

例

fgetpos() を参照してください。

出力

関連項目

ftell

構文

#include <**stdio.h**>
long int ftell(FILE **stream*);

目的

ストリームのファイル位置インジケータの現在の値を取得します。

戻り値

成功すると、 **ftell** 関数はストリームのファイル位置インジケータの現在の値を返します。失敗すると、 **ftell** 関数は-1L を返し、**errno** 内に実装で定義される正の値を格納します。

パラメータ

stream ストリームへのポインタ。

説明

ftell 関数は *stream* が指すストリームのファイル位置インジケータの現在の値を取得します。バイナリストリームでは、ファイル位置インジケータの値はファイルの先頭からの文字数です。テキストストリームの場合、ファイル位置インジケータには未指定の情報が含まれます。その情報は、 **fseek** 関数でストリームのファイル位置インジケータを **ftell** 呼び出し時点の位置に戻すために使用できます。これら 2 つの戻り値の差異は、読み書きされた文字数を測定する基準として必ずしも有効ではありません。

例

```
/* a sample program that reads the third charter
in a file. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *stream;
    char ch;
    int cur_indicator;
    long int offset = 2*sizeof(char);
    /* open file for reading and writing */
    if((stream = fopen("ftell.c","r")) == NULL){
      printf("cannot open file\n");
      exit(1);
    }
    fseek(stream,offset,0);     //search the third character
    cur_indicator = ftell(stream);      // read current indicator
    fread(&ch,sizeof(char),1,stream);   //read a character
    printf("The third character in file is '%c'\n",ch);
    fseek(stream,cur_indicator-1,0);   // set the file indicator to
                                       // the number before the current
    fread(&ch,sizeof(char),1,stream);   //read a character
```

```
    printf("The second character in file is '%c'\n",ch);
    rewind(stream);                     //set file indicator to first
    fread(&ch,sizeof(char),1,stream);
    printf("The first character in file is '%c'\n",ch);
    fclose(stream);
}
```

出力

```
The third character in file is ' '
The second character in file is '*'
The first character in file is '/'
```

関連項目

fwrite

構文

#include <**stdio.h**>
int fwrite(void * *restrict ptr*, **size_t** *size*, **size_t** *nmemb*, **FILE** * *restrict stream*);

目的

ストリームにデータを書き込みます。

戻り値

fwrite 関数は書き込みに成功した要素数を返します。この要素数は、書き込みエラーが発生した場合、*nmemb* 未満である可能性があります。*size* または *nmemb* がゼロである場合、*fwrite* はゼロを返します。

パラメータ

ptr 書き込む文字列を格納した配列へのポインタ。

size バイト単位の要素のサイズ。

nmemb 書き込む最大要素数。

stream ストリームへのポインタ。

説明

fwrite 関数は、*size* で指定されたサイズの要素を最大数 *nmemb* まで、*ptr* が指す配列から *stream* が指すストリームに書き込みます。ストリームのファイル位置インジケータ (定義されている場合) は、書き込みに成功した文字数の分だけ進められます。エラーが発生した場合、ストリームのファイル位置インジケータの結果の値は不定です。

例

```
/* a sample program that flushes the buffer
after write operation. */
#include <stdio.h>
#include <stdlib.h>
#define SIZE 80

int main() {
    FILE *stream;
    char buf[SIZE];
    char *name;
    size_t num;

    if((name = tmpnam(NULL)) == NULL)
        printf("Error in creating temp file.\n");
    /* open file for reading and writing */
    if((stream = fopen(name,"w")) == NULL) {
        printf("cannot open file\n");
```

```
        exit(1);
    }
    strcpy(buf,"1111111111111111");
    num = fwrite(buf,sizeof(char),10,stream);
    printf("Number of elements written to temp file: %d\n", num);
    fflush(stream);      // flush the buffer
    fclose(stream);
    remove(name);
}
```

出力

```
Number of elements written to temp file: 10
```

関連項目

getchar

構文

#include <**stdio.h**>
int getchar(void);

目的

標準入力ストリームから次の文字を読み込みます。

戻り値

getchar 関数は **stdin** が指す入力ストリームの次の文字を返します。ストリームがファイルの終わりにある場合、ストリームの EOF インジケータが設定され、**getchar** は **EOF** を返します。読み込みエラーが発生した場合は、ストリームのエラーインジケータが設定され、 **getchar** は **EOF** を返します。

パラメータ

引数はありません。

説明

getchar 関数は引数 **stdin** を入力ストリームに指定した **getc** と同等です。

例

出力

関連項目

getc

構文

#include <**stdio.h**>
int getc(**FILE** **stream*);

目的

ストリームから入力を読み込みます。

戻り値

getc 関数は *stream* が指す入力ストリームの次の文字を返します。ストリームがファイルの終わりにある場合、ストリームの EOF 終わりのインジケータが設定され、 **getc** は **EOF** を返します。読み込みエラーが発生した場合は、ストリームのエラーインジケータが設定され、 **getc** は **EOF** を返します。

パラメータ

stream ストリームへのポインタ。

説明

getc 関数は **fgetc** と同等ですが、マクロとして実装される場合は、*stream* を複数回評価することがあるため、*stream* 引数を副作用のない式にする必要があります。

例

```
/* a sample program that flushes the buffer
after write operation. */
#include <stdio.h>
#include <stdlib.h>

int main(int argc , char *argv[]) {
    char choice;

    do {
      printf("1: Check spelling\n");
      printf("2: Correct spelling\n");
      printf("3: Lookup a word in the directory\n");
      printf("4: Quit\n");

      printf("\nEnter your selection: ");
      choice = getc(stdin);
    } while (!strchr("1234",choice));
    printf("You entered: %c\n", choice);
    return (0);
}
```

出力

```
1: Check spelling
2: Correct spelling
3: Lookup a word in the directory
```

```
4: Quit

Enter your selection: You entered: 4
```

関連項目

gets

構文

#include <**stdio.h**>
int ***gets**(**char** **s*);

目的

入力ストリームの文字を配列に読み込みます。

戻り値

成功すると、 **gets** 関数は *s* を返します。ファイルの終わりに到達し、配列に文字が読み込まれなかった場合、配列の内容は変更されずに null ポインタが返されます。操作中に読み込みエラーが発生した場合、配列の内容は不定となり null ポインタが返されます。

パラメータ

s ストリーム内の文字の読み込み先配列へのポインタ。

説明

gets 関数は、ファイルの終わりに到達するか、または改行文字が読み込まれるまで、*stdin* が指す入力ストリームの文字を *s* が指す配列に読み込みます。改行文字がある場合は破棄され、配列に最後の文字が読み込まれた直後に null 文字が書き込まれます。

例

出力

関連項目

perror

構文

#include <**stdio.h**>
void perror(const char **s**);

目的

errno 内のエラー番号をエラーメッセージにマップします。

戻り値

perror 関数は値を返しません。

パラメータ

s 表示するメッセージへのポインタ。

説明

perror 関数は整数式 **errno** のエラー番号をエラーメッセージにマップします。標準エラーストリームに文字のシーケンスを次のように書き込みます。最初に (*s* が null ポインタではなく *s* が指す文字が null 文字ではない場合) 後ろにコロンと空白 1 つが付く s が指す文字列、次に、後ろに改行文字の付く適切なエラーメッセージ文字列。エラーメッセージ文字列の内容は、引数 **errno** を使用する **strerror** 関数が返す内容と同じです。

例

```
/* Uses the perror function to display an error message.  */
#include <stdio.h>
#include <signal.h>
int delay = 1;
void childHandler();

int main()
{
   int pid;
   char *argv[3];

   argv[0]="-l";
   signal(SIGCHLD, childHandler);
   pid=fork();
   if(pid==0)
   {
      execvp("i",argv);
      perror("Error:");
   }
   else
   {
      sleep(delay);
      printf("Child %d exceeded limit and is being killed \n",pid);
      kill(pid, SIGINT);
   }
```

```
}

void childHandler()
{
   int childPid, childStatus;
   childPid=wait(&childStatus);
   printf("Child %d terminated within %d seconds \n", childPid, delay);
   exit(0);
}
```

出力

```
Error:: No such file or directory
Child 10698 terminated within 1 seconds
```

関連項目

printf

構文

#include <**stdio.h**>
int printf(**const char** *restrict format*, ...);

目的

stdout に出力を行います。

戻り値

printf 関数は、送信された文字数を返すか、出力エラーまたはエンコーディングエラーが発生した場合は負の値を返します。

パラメータ

format 書式文字列へのポインタ。

説明

printf 関数は、 **stdout** を出力先ストリームに指定した **fprintf** と同等です。

例

```
/* this program prints a string.
*/

#include <stdio.h>

int main() {
    char *c = "This is a test string";
    printf("%s\n", c);
}
```

出力

```
This is a test string
```

関連項目

putchar

構文

#include <**stdio.h**>
int putchar(int *c*);

目的

次の文字を **stdout** に書き込みます。

戻り値

putchar 関数は書き込まれた文字を返します。書き込みエラーが発生した場合は、ストリームのエラーインジケータが設定され、 **putchar** は **EOF** を返します。

パラメータ

c 表示する文字。

説明

putchar 関数は引数 **stdout** を出力先ストリームに指定した **putc** と同等です。

例

```
/* a sample program that writes a string
to stdout. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    char str[] = "This is a test.\n";
    char *p;

    p = str;
    for(; *p; p++) putchar(*p);
}
```

出力

```
This is a test.
```

関連項目

putc

構文

#include <**stdio.h**>
int putc(int *c*, **FILE** **stream*);

目的

ストリームに出力を行います。

戻り値

putc 関数は書き込まれた文字を返します。書き込みエラーが発生した場合は、ストリームのエラーインジケータが設定され、 **putc** は **EOF** を返します。

パラメータ

c 表示する文字。

stream ストリームへのポインタ。

説明

putc 関数は **fputc** と同等ですが、マクロとして実装される場合は、stream を複数回評価することがあるため、*stream* 引数を副作用のない式にする必要があります。

例

```
/* a sample program that writes a string
to a file. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *stream;
    char *name;
    int c, i;

    c = 'a';
    if((name = tmpnam(NULL)) == NULL)
        printf("temp file name could not be made. \n");
    /* open file for reading and writing */
    if((stream = fopen(name,"w")) == NULL) {
        printf("cannot open file\n");
        exit(1);
    }
    for(i =0; i <10; i++) {
        putc(c, stream);
        printf("%c", (char)c);
    }
    printf("\n");
    fclose(stream);
    remove(name);
}
```

出力

```
aaaaaaaaaa
```

関連項目

puts

構文

#include <**stdio.h**>
int puts(const char ******s***);**

目的

文字列を出力します。

戻り値

書き込みエラーが発生した場合、 **puts** 関数は **EOF** を返します。それ以外の場合は、負ではない値を返します。

パラメータ

s 書き込む文字列へのポインタ。

説明

puts 関数は、*s* が指す文字列を **stdout** が指すストリームに書き込み、改行文字を出力に追加します。終端の null 文字は書き込まれません。

例

```
/* a sample program that writes a string
to stdout. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    char str[80];

    strcpy(str,"this is a test.\n");
    puts(str);
}
```

出力

```
this is a test.
```

関連項目

remove

構文

#include <**stdio.h**>
int remove(**const char** **filename*);

目的

ファイルを削除します。

戻り値

remove 関数は、操作が成功した場合はゼロを、失敗した場合はゼロ以外を返します。

パラメータ

filename ファイル名を表す文字列へのポインタ。

説明

remove 関数は、名前が *filename* が指す文字列であるファイルにその名前でアクセスできないようにします。その名前で新しいファイルを作成していない限り、以降にその名前を使用してファイルを開こうとしても失敗します。ファイルが開いている場合の **remove** 関数の動作は、実装で定義されます。

例

出力

関連項目

rename

構文

#include <**stdio.h**>
int rename(const char **old*, **const char** **new*);

戻り値

ファイル名を変更します。

戻り値

rename 関数は、操作が成功した場合はゼロを返します。失敗した場合は非ゼロを返し、ファイルが既に存在していたときは元の名前のままになります。

パラメータ

old 古いファイル名を表す文字列へのポインタ。

new 新しいファイル名を表す文字列へのポインタ。

説明

rename 関数は、*old* が指す文字列の名前を持つファイルを、以降は *new* が指す文字列に指定した名前で認識されるようにします。*old* の名前のファイルは、その名前ではアクセスできなくなります。*new* が指す文字列の名前を持つファイルが **rename** 関数を呼び出す前から存在している場合、動作は実装で定義されます。

例

```
/* a sample program that renames the file specified as the first
command line argument to that specified by the second command
line argument. this program rename.c is executed by command
       rename.c < rename.in
   Input file rename.in contains the following data: rename.c temp
*/
#include <stdio.h>

int main() {
    char name1[80], name2[80];

    printf("Enter filename: ");
    scanf("%s", name1);
    printf("%s\n", name1);
    printf("Enter new name: ");
    scanf("%s", name2);
    printf("%s\n", name2);
    if(rename(name1, name2)!=0)
        printf("Rename error\n");
    else {
printf("Old file name: %s\n",name1);
printf("New file name: %s\n", name2);
```

```
    }
    if(rename(name2, name1) !=0)
        printf("Rename error\n");
    else {
printf("Old file name: %s\n", name2);
printf("New file name: %s\n", name1);
    }
    return (0);
}
```

出力

```
Enter filename: rename.c
Enter new name: temp
Old file name: rename.c
New file name: temp
Old file name: temp
New file name: rename.c
```

関連項目

rewind

構文

#include <**stdio.h**>
void rewind(FILE **stream*);

目的

ストリームの先頭にファイル位置インジケータを位置付けます。

戻り値

rewind 関数は値を返しません。

パラメータ

stream ストリームへのポインタ。

説明

rewind 関数は、*stream* が指すストリームのファイル位置インジケータをファイルの先頭に位置付けます。以下と同等です。

(void)fseek(*stream, 0L, SEEK_SET***)**

ただし、ストリームのエラーインジケータもクリアされる点が異なります。

例

fseek() を参照してください。

出力

関連項目

scanf

構文

#include <**stdio.h**>
int scanf(const char *restrict format, ...*);

目的

stdin からの入力をスキャンします。

戻り値

scanf 関数は、変換前に入力エラーが発生した場合、マクロ **EOF** の値を返します。それ以外の場合は、代入された入力項目の数を返します。以前にマッチングエラーが発生している場合、この数は指定された入力項目の数よりも少ない可能性があり、ゼロになることもあります。

パラメータ

format 書式文字列へのポインタ。

説明

scanf 関数は、 **stdin** を入力ストリームに指定した **fscanf** と同等です。

例

```
/* this program scanf.c is executed by command
       scanf.c < scanf.in
   Input file scanf.in contains the following data
       1.0 2.0 3.0 4.0
*/
#include <stdio.h>

int main() {
    float f;
    double d;

    printf("input f\n");
    scanf("%f",&f);
    printf("Your input of f is %f\n", f);
    printf("input d\n");
    scanf("%lf",&d);
    printf("Your input of d is %f\n", d);
    printf("input f\n");
    scanf("%lf",&f); /* ERROR: should use %f for float */
    printf("Your input of d is not what you typed in f = %f\n", f);
    printf("input d\n");
    scanf("%f",&d); /* ERROR: should use %lf for double */
    printf("Your input of d is not what you typed in d = %f\n", d);
}
```

出力

```
input f
Your input of f is 1.000000
input d
Your input of d is 2.000000
input f
Your input of d is not what you typed in f = 2.125000
input d
Your input of d is not what you typed in d = 512.000000
```

関連項目

setbuf

構文

#include <**stdio.h**>
void setbuf(FILE **restrict stream*, **char** * *restrict buf*);

目的

ストリームが使用するバッファを指定します。

戻り値

setbuf 関数は値を返しません。

パラメータ

stream ストリームへのポインタ。

buf 使用するバッファへのポインタ。

説明

値を返さないという点を除き、 **setbuf** 関数は **setvbuf** 関数と同等であり、*mode* の値として **_IOFBF** および *size* の値として **BUFSIZ** を使用して呼び出されるか、(*buf* が null ポインタである場合は) *mode* の値として **_IONBF** を使用して呼び出されます。

例

```
/* a sample program that reads a file then
displays it's content. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *stream;
    char buffer[BUFSIZ], ch;

    /* open file for reading only */
    if((stream=fopen("setbuf.c","r")) == NULL) {
        printf("cannot open file\n");
        exit(1);
    }
    setbuf(stream,buffer); /* associate a buffer with the stream */
    ch = fgetc(stream);
    printf("%c\n",ch);
    fclose(stream);
}
```

出力

```
/
```

関連項目

setvbuf

構文

#include <**stdio.h**>
int setvbuf(FILE * *restrict stream*, **char** * *restrict buf*, **int** *mode*, **size_t** *size*);

目的

特定のストリームのモードとバッファサイズを指定します。

戻り値

setvbuf 関数は、成功すると 0 を返します。モードの値が無効か、または要求を満たすことができない場合は、0 以外の値を返します。

パラメータ

stream ストリームへのポインタ。

buf 使用するバッファへのポインタ。

mode ストリームをバッファリングする方法示す整数値。詳細については下の [説明] を参照してください。

size バッファのサイズ。

説明

setvbuf 関数を使用できるのは、*stream* が指すストリームを開いた状態のファイルに関連付けた後、**setvbuf** への関数呼び出しの失敗以外のストリーム上で他の操作を実行するまでの間です。引数 *mode* はストリームをバッファする方法を次のように指定します。**_IOLBF** は入出力をフルバッファリングします。**_IOFBF** は入出力をラインバッファリングします。**_IONBF** は入出力をバッファリングしません。*buf* が null ポインタでない場合、*buf* が指す配列を **setvbuf** 関数が割り当てるバッファの代わりに使用でき、引数 *size* で配列のサイズを指定します。その他の場合は、 **setvbuf** 関数が割り当てるバッファのサイズを *size* で決定できます。配列の内容は常に不定です。

例

```
/* a sample program that reads a file then
displays it's content. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *stream;
    char buffer[500], ch;
    size_t size = 500;

    /* open file for reading only */
    if((stream=fopen("setvbuf.c","r")) == NULL){
```

```
      printf("cannot open file\n");
      exit(1);
    }
    setvbuf(stream,buffer,_IOFBF,size); /* associate a buffer with the stream */
    ch = fgetc(stream);
    printf("%c\n",ch);
    fclose(stream);
}
```

出力

```
/
```

関連項目

snprintf

構文

#include <**stdio.h**>
int snprintf(**char** * *restrict s*, **size_t** *n*, **const char** * *restrict format*, ...);

目的

配列に出力を行います。

戻り値

snprintf 関数は、**n** が十分な大きさであれば書き込まれていたはずの文字数を返します。この数には終端の null 文字は含まれません。または、エンコーディングエラーが発生した場合は負の値を返します。したがって、null で終わる出力は、戻り値が負でなくかつ **n** より小さい場合に限り完全に書き込まれています。

パラメータ

s 書式設定された文字列を格納する配列へのポインタ。

n *s* に格納可能な最大文字数。

format 書式文字列へのポインタ。

説明

snprintf 関数は、出力がストリームではなく配列 (引数 *s* で指定される) に書き込まれるという点を除き、**fprintf** と同じです。*n* がゼロである場合、何も書き込まれず、*s* が null ポインタであっても構いません。それ以外の場合、**n**-1 番目より後の出力文字は配列に書き込まれず、破棄されます。重複するオブジェクト間でコピーが実行される場合、動作は不定です。

例

出力

関連項目

sprintf

構文

#include <**stdio.h**>
int sprintf(char * *restrict s,* **const char** * *restrict format, ...*);

目的

配列に出力を行います。

戻り値

sprintf 関数は、終端の null 文字をカウントに含めずに、配列に書き込まれる文字数を返します。また、エンコーディングエラーが発生した場合は負の値を返します。

戻り値

s 書式設定された文字列を格納する配列へのポインタ。

format 書式文字列へのポインタ。

説明

sprintf 関数は、出力がストリームではなく配列 (引数 **s** で指定される) に書き込まれるという点を除き、 **fprintf** と同じです。書き込まれた文字の終端には null 文字が記述されますが、戻り値の一部としてはカウントされません。重複するオブジェクト間でコピーが実行される場合、動作は不定です。

例

出力

関連項目

sscanf

構文

#include <**stdio.h**>
int sscanf(const char *s,* **const char** * *restrict format, ...*);

目的

書式化された文字列から入力を読み込みます。

戻り値

sscanf 関数は、変換前に入力エラーが発生した場合、マクロ **EOF** の値を返します。それ以外の場合は、代入された入力項目の数を返します。以前にマッチングエラーが発生している場合、この数は指定された入力項目の数よりも少ない可能性があり、ゼロになることもあります。

パラメータ

s 書式化された文字列を格納している配列へのポインタ。

format 書式文字列へのポインタ。

説明

sscanf 関数は、入力がストリームからではなく文字列 (引数 **s** で指定される) から取得されるという点を除き、 **fscanf** と同じです。文字列の終わりに到達することは、 **fscanf** 関数でファイルの終わりに到達することと同じです。重複するオブジェクト間でコピーが実行される場合、動作は不定です。

例

出力

関連項目

tmpfile

構文

#include <stdio.h>
FILE *tmpfile(void);

目的

一時ファイルを作成します。

戻り値

tmpfile 関数では作成したファイルのストリームへのポインタを返します。ファイルを作成できなかった場合、 **tmpfile** 関数は null ポインタを返します。

パラメータ

引数はありません。

説明

tmpfile 関数は、ファイルが閉じられるときまたはプログラムの終了時に自動的に削除される一時ファイルを作成します。プログラムが異常終了した場合に、開いている一時ファイルが削除されるかどうかは、実装で定義されます。ファイルは**"wb+"**の更新モードで開かれます。

例

```
/* a sample program that creates a temporary working file. */
#include <stdlib.h>
#include <stdio.h>

int main() {
    FILE *temp;

    if((temp = tmpfile()) == NULL) {
        printf("Cannot open temporary work file.\n");
        exit(1);
    }
}
```

出力

関連項目

tmpnam

構文

#include <**stdio.h**>
char ***tmpnam**(**char** **s*);

目的

ファイル名を示す一時的な文字列を作成します。

戻り値

引数が null ポインタである場合、 **tmpnam** 関数は内部の静的オブジェクトに結果を残し、そのオブジェクトへのポインタを返します。 **tmpnam** 関数の後続の呼び出しにより、同一のオブジェクトが変更される場合があります。引数が null ポインタでない場合、引数は少なくとも **L_tmpnam char** の配列を指すと見なされます。 **tmpnam** 関数はその配列を結果に書き込み、引数を値として返します。

パラメータ

s 一時ファイル名を表す文字列を格納する配列へのポインタ。

説明

tmpnam 関数は、有効なファイル名であり、既存のファイル名と同じではない文字列を生成します。

tmpnam 関数は、呼び出されるたびに (最大 **TMP_MAX** 回) 異なる文字列を生成します。 **TMP_MAX** 回より多く呼び出された場合の動作は、実装で定義されます。

実装では、ライブラリ関数による **tmpnam** 関数の呼び出しが存在しないかのように動作する必要があります。

例

```
/* a sample program that displays three unique temporary
file name. */
#include <stdio.h>

int main() {
    char name[40];
    int i;

    for(i = 0; i < 3; i++) {
      tmpnam(name);
      printf("%s\n",name);
    }
}
```

出力

```
/var/tmp/aaaxraG7u
/var/tmp/baayraG7u
/var/tmp/caazraG7u
```

関連項目

ungetc

構文

#include <**stdio.h**>
int ungetc(int *c*, **FILE** **stream*);

目的

入力ストリームに文字を押し戻します。

戻り値

ungetc 関数は変換後に押し戻された次の文字を返します。または、操作が失敗した場合は **EOF** を返します。

パラメータ

c 押し戻す文字。

stream ストリームへのポインタ。

説明

ungetc 関数は、*c*(**unsigned char** に変換される) で指定される文字を *stream* が指す入力ストリームに押し戻します。そのストリームに対する後続の読み込みでは、押し戻された文字が、押し戻された順とは逆の順に返されます。**fseek**、**fsetpos**、または **rewind** のファイル位置決め関数の呼び出し (*stream* が指すストリームを使用) の介在が成功すると、ストリームに押し戻された文字はすべて破棄されます。ストリームに対応する外部記憶域は変更されません。

1 文字の押し戻しが保証されます。同じストリームに対して **ungetc** 関数を何度も呼び出し、そのストリームの読み取りまたはファイル位置決め操作を間にはさんでいない場合は、操作が失敗する可能性があります。

c の値がマクロ **EOF** の値と同じである場合、操作は失敗し、入力ストリームは変更されません。

ungetc 関数の呼び出しが成功すると、ストリームの EOF インジケータがクリアされます。読み込みまたは破棄後のストリームのファイル位置インジケータの値は、文字が押し戻される前と同じです。テキストストリームでは、 **ungetc** 関数の呼び出しが成功した後のファイル位置インジケータの値は、押し戻されたすべての文字が読み取られるか破棄されるまで、未指定です。バイナリストリームでは、**ungetc** 関数の呼び出しが成功するたびにファイル位置インジケータがデクリメントされます。呼び出し前にファイル位置インジケータの値がゼロであった場合、呼び出し後の値は不定です。

例

```
/* a sample program that inserts a character into the
buffer of a file then reads it and displays it. */
#include <stdio.h>
```

```
#include <stdlib.h>
#include <fcntl.h>
#define BUF_SIZE 40


int main() {
    char buf[BUF_SIZE], c;
    FILE *stream;

    if((stream = fopen("ungetc.c", "r")) != NULL) {
        c = fgetc(stream);
        printf("The character pulled from the file is: %c\n", c);
        printf("ungetc(c,stream) = ");
        fputc( ungetc(c,stream), stdout);
        printf("\n");
    }
    else
        printf("Error in opening file!\n");
    fclose(stream);
}
```

出力

```
The character pulled from the file is: /
ungetc(c,stream) = /
```

関連項目

vfprintf

構文

#include <**stdio.h**>
int vfprintf(FILE * *restrict stream*, **const char** * *restrict format*, **va_list** *arg*);

目的

特定のストリームに出力を行います。

戻り値

vfprintf 関数は、送信された文字数を返すか、出力エラーまたはエンコーディングエラーが発生した場合は負の値を返します。

パラメータ

stream ストリームへのポインタ。

format 書式文字列へのポインタ。

arg 引数リストへのポインタ。

説明

vfprintf 関数は、*arg* によって置換される可変長の引数リストを使用する **fprintf** と同じです。これは、**va_start** マクロ (および、たぶん引き続く **va_arg** 呼び出し) によって初期化されている必要があります。 **vfprintf** 関数は **va_end** マクロを呼び出しません。

例

出力

関連項目

vprintf

構文

#include <**stdio.h**>
int vprintf(**const char** * *restrict format*, **va_list** *arg*);

目的

特定のストリームに出力を行います。

戻り値

vprintf 関数は、送信された文字数を返すか、出力エラーまたはエンコーディングエラーが発生した場合は負の値を返します。

パラメータ

format 書式文字列へのポインタ。

arg 引数リストへのポインタ。

説明

vprintf 関数は、*arg* によって置換される可変長の引数リストを使用する **printf** と同じです。これは、**va_start** マクロ (および、たぶん引き続く **va_arg** を呼び出し) によって初期化されている必要があります。 **vprintf** 関数は **va_end** マクロを呼び出しません。

例

出力

関連項目

vsnprintf

構文

#include <**stdio.h**>
int vsnprintf(**char** * *restrict s*, **size_t** *n*, **const char** * *restrict format*, **va_list** *arg*);

目的

配列に出力を行います。

戻り値

vsnprintf 関数は、*n* が十分な大きさであれば書き込まれていたはずの文字数を返します。この数には終端の null 文字は含まれません。または、エンコーディングエラーが発生した場合は負の値を返します。したがって、null で終わる出力は、戻り値が負でなくかつ *n* より小さい場合に限り完全に書き込まれています。

パラメータ

s 書式設定された文字列を格納する配列へのポインタ。

n *s* に格納可能な最大文字数。

format 書式文字列へのポインタ。

arg 引数リストへのポインタ。

説明

vsnprintf 関数は、*arg* によって置換される可変長の引数リストを使用する **snprintf** と同じです。これは、 **va_start** マクロ (および、たぶん引き続く **va_arg** を呼び出し) によって初期化されている必要があります。 **vsnprintf** 関数は **va_end** マクロを呼び出しません。重複するオブジェクト間でコピーが実行される場合、動作は不定です。

関連項目

vsprintf

構文

#include <**stdio.h**>
int vsprintf(char * *restrict s*, **const char** * *restrict format*, **va_list** *arg*);

目的

配列に出力を行います。

戻り値

vsprintf 関数は、*n* が十分な大きさであれば書き込まれていたはずの文字数を返します。この数には終端の null 文字は含まれません。または、エンコーディングエラーが発生した場合は負の値を返します。

パラメータ

s 文字へのポインタ。

format 文字へのポインタ。

arg 引数。

説明

vsprintf 関数は、*arg* によって置換される可変長の引数リストを使用する **sprintf** と同じです。これは、**va_start** マクロ (および、たぶん引き続く **va_arg** を呼び出し) によって初期化されている必要があります。 **vsprintf** 関数は **va_end** マクロを呼び出しません。重複するオブジェクト間でコピーが実行される場合、動作は不定です。

例

出力

関連項目

第18章　汎用標準関数 — <stdlib.h>

ヘッダー **stdlib.h** には、汎用ユーティリティの５つの型と複数の関数が宣言され、複数のマクロが定義されています。

宣言されている型は、 **size_t** および **wchar_t** です。

div_t

この構造体型は **div** 関数の戻り値の型です。

ldiv_t

この構造体型は **ldiv** 関数の戻り値の型です。

定義されているマクロは次のとおりです。
EXIT_FAILURE

および

EXIT_SUCCESS

これらは、 **exit** 関数の引数として使用できる定数式に展開され、それぞれ失敗および成功の終了状態をホスト環境に返します。

RAND_MAX

これは整数の定数式に展開され、その値は **rand** 関数から返される最大値です。

MB_CUR_MAX

これは **size_t** 型の正の整数式に展開されます。その値は、現在のロケール (カテゴリ **LC_TYPE**) によって指定される拡張文字セットのマルチバイト文字の最大バイト数であり、必ず **MB_LEN_MAX** 以下となります。

パブリックデータ
なし。

関数

次の関数がヘッダーファイル **stdlib.h** でプロトタイプ化されています。

関数	説明
abort()	プログラムを異常終了します。
abs()	絶対値を計算します。
atexit()	プログラム終了時にコールされる関数を登録します。
atoc()	文字列の一部を double complex 型の表現に変換します。
atof()	文字列の一部を double 型の表現に変換します。
atoi()	文字列の一部を int 型の表現に変換します。
atol()	文字列の一部を long int 型の表現に変換します。
atoll()	文字列の一部を long long int 型の表現に変換します。
bsearch()	一致するオブジェクトを検索します。
calloc()	配列に対して領域を割り当てます。
div()	商および剰余を計算します。
exit()	プログラムを正常に終了します。
free()	領域の割り当てを解除します。
getenv()	環境変数に一致する文字列を検索します。
iscnum()	文字列が有効な複素数であるかどうかを調べます。
isenv()	文字列が環境変数であるかどうかを調べます。
isnum()	文字列が有効な数値であるかどうかを調べます。
iswnum()	ワイド文字の文字列が有効な数値であるかどうかを調べます。
labs()	絶対値を計算します。
malloc()	オブジェクトに対して領域を割り当てます。
mblen()	マルチバイト文字のバイト数を特定します。
mbstowcs()	マルチバイト文字のシーケンスを対応するコードに変換します。
mbtowc()	マルチバイト文字をワイド文字に変換します。
qsort()	オブジェクトの配列を並べ替えます。
rand()	擬似乱数の整数を返します。
realloc()	メモリブロックの再割り当てを行ないます。
remenv()	環境変数を削除します。
srand()	新規の擬似乱数の整数を返します。
strtod()	文字列の最初の部分を double 型に変換します。
strtol()	文字列の最初の部分を long 型に変換します。
strtoll()	文字列の最初の部分を long long int 型に変換します。
strtoul()	文字列の最初の部分を符号なし long 型に変換します。
system()	コマンドプロセッサを使用してプログラムを実行します。
wcstombs()	コードのシーケンスを対応するマルチバイト文字に変換します。
wctomb()	ワイド文字をマルチバイト文字に変換します。

マクロ

stdlib ヘッダーには、次のマクロが定義されています。

マクロ	説明
EXIT_FAILURE	失敗の終了状態を返します。
EXIT_SUCCESS	成功の終了状態を返します。
RAND_MAX	**rand** 関数から返される最大値を保持する整数。
MB_CUR_MAX	現在のロケールによって指定された最大バイト数を保持する符号なし整数。

宣言された型

stdlib ヘッダーには次の型が宣言されています。

宣言された型	説明
size_t	符号なし整数。
wchar_t	整数。
div_t	**div** から返される構造体の型。
ldiv_t	**ldiv** から返される構造体の型。

移植性

このヘッダーには移植性に関する既知の問題はありません。

abort

構文

#include <**stdlib.h**>
void abort(*void***);**

目的

プログラムを異常終了します。

戻り値

関数 **abort()** は呼び出し元に戻りません。

パラメータ

引数はありません。

説明

abort() 関数はプログラムの異常終了を発生させます。ただし、シグナル **SIGABRT** が捕捉されてシグナルハンドラが戻らない場合はこの限りではありません。開いた状態の出力ストリームをフラッシュするか、開いた状態のストリームを閉じるか、または一時ファイルを削除するかは、実装で定義されます。正常でない終了状態は、関数呼び出し **raise(SIGABRT)** によって、実装で定義された形式でホスト環境に返されます。

例

```
/* a sample program that use abort()*/
#include <stdlib.h>
#include <stdio.h>

int main() {
    printf("abort() is called\n");
    abort();
    printf("This is not called\n");
}
```

出力

```
abort() is called
```

関連項目

abs

構文

#include <**stdlib.h**>
int abs(int *j***);**
long int labs(long int *j***);**

目的

絶対値を計算します。

戻り値

abs() および **labs()** 関数は絶対値を返します。

パラメータ

j 整数の引数。

説明

abs() および **labs()** 関数は、整数 *j* の絶対値を計算します。結果を表現できない場合、動作は不定です。

例

```
/* A sample program to compute absolute values.*/
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num = -2;
    printf("Number is %d\n", num);
    printf("but after running abs()\n");
    num = abs(num);
    printf("Number is %d\n", num);
}
```

出力

関連項目

atexit

構文

#include <**stdlib.h**>
int atexit(void (*func)(*void***));**

目的

プログラム終了時にコールされる関数を登録します。

戻り値

atexit() 関数は、登録が成功した場合はゼロを、失敗した場合はゼロ以外を返します。

パラメータ

func プログラムの正常終了時に呼び出される登録関数。

説明

atexit() 関数は、*func* が指す関数を登録して、プログラムの正常終了時に引数なしで呼び出されるようにします。

環境制限

登録できる関数の数に制限はありません。

例

```
/* this is a sample that use atexit() */
#include <stdio.h>
#include <stdlib.h>
void atexitHandler();

int main()
{
    printf("The program is running. \n");
    atexit(atexitHandler);   /* call atexitHandler before exit */
}

void atexitHandler()
{
    printf("Call function to exit. \n ");
}
```

出力

```
The program is running.
Call function to exit.
```

関連項目
exit()

atoc

構文

#include <**stdlib.h**>
double complex atoc(const char **nptr***);*

目的

文字列の一部を double complex 型の表現に変換します。

戻り値

atoc() 関数は変換した値を返します。

パラメータ

nptr double complex 型数値に変換する文字へのポインタ。

説明

atoc() 関数は、*nptr* が指す文字列の最初の部分を **double complex** 型の表現に変換します。複素数は、complex(re, im)、または、a、$a+ib$、$a+i*b$、$ib+a$、$i*b$ などの従来の数学的表記として表すことができます。ここで、虚数 i は j または I に置換できます。また、a および b は定数であり、正の符号'+' は負の符号'-' に置換できます。

例

```
/* a sample program that changes a string to double complex*/
#include <stdlib.h>
#include <complex.h>

int main() {
    char *number = "1.27859-I*32.6532";
    char *number2 = "1.27859+i32.6532";
    char *number3 = "complex(3,4)";
    double complex outnumber;

    outnumber = atoc(number);
    printf("the complex number is %f \n",outnumber);
    outnumber = atoc(number2);
    printf("the complex number is %f \n",outnumber);
    outnumber = atoc(number3);
    printf("the complex number is %f \n",outnumber);
}
```

出力

```
the complex number is complex(1.278590,-32.653200)
the complex number is complex(1.278590,32.653200)
the complex number is complex(3.000000,4.000000)
```

関連項目

atof(), **atoi**(), **strtod**(), **strtof**(), **strtold**().

atof

構文

#include <**stdlib.h**>
double atof(const char ******nptr***);**

目的

文字列の一部を double 型の表現に変換します。

戻り値

atof() 関数は変換した値を返します。

パラメータ

nptr double 型数値に変換する文字へのポインタ。

説明

atof() 関数は、*nptr* が指す文字列の最初の部分を **double** 型の表現に変換します。エラー時の動作以外は、次の式と同じです。

strtod(nptr, (char **)NULL)

例

```
/* a sample program that changes a string to double */
#include <stdlib.h>

int main() {
    char *number = "1.27859";
    double outnumber;

    outnumber = atof(number);
    printf("the out double number is %f \n",outnumber);
}
```

出力

```
the out double number is 1.278590
```

関連項目

atoc(), **atoi**(), **strtod**(), **strtof**(), **strtold**().

atoi

構文

#include <**stdlib.h**>
int atoi(const char ******nptr***);**
long int atol(const char ******nptr***);**
long long int atoll(const char ******nptr***);**

目的

文字列の一部を int、long int、または long long int 型の表現に変換します。

戻り値

atoi() 、 **atol**() 、および **atoll**() 関数は変換した値を返します。

パラメータ

nptr int、long int、または long long int 型数値に変換する文字へのポインタ。

説明

atoi() 、 **atol**() 、および **atoll**() 関数は、*nptr* が指す文字列の最初の部分を **int**、**long int**、および **long long int** 型の表現にそれぞれ変換します。エラー時の動作以外は、次の式と同じです。

atoi(): (int)strtol(nptr, (char **)NULL, 10)

atol(): strtol(nptr, (char **)NULL, 10)

atoll(): strtoll(nptr, (char **)NULL, 10)

例

```
/* a sample program that changes a string to integer */
#include <stdlib.h>

int main() {
    char *number = "1278";
    int outnumber;

    outnumber = atoi(number);
    printf("the out integer number is %d \n",outnumber);
}
```

出力

```
the out integer number is 1278
```

関連項目

atoc(), **atof**(), **strtol**(), **strtoll**(), **strtoul**(), **strtoull**().

bsearch

構文

#include <**stdlib.h**>
void * bsearch(const void ******key***, const void** ******base***, size_t** *nmemb***, size_t** *size***, int (****compar***)(const void *, const void *));**

目的

一致するオブジェクトを検索します。

戻り値

bsearch() 関数は、一致する配列要素へのポインタを返し、一致が見つからなかった場合は null ポインタを返します。2 つの要素が等しいと判定された場合、どちらの要素を一致とするかは未指定です。

パラメータ

key void へのポインタ。
base void へのポインタ。
nmemb 配列の要素数を表す整数型の引数。
size 配列の各要素のバイト数を表す整数型の引数。
compar 2 つの引数と int 型の戻り値を持つ関数へのポインタ。

説明

bsearch() 関数は、*base* が最初の要素を指す、要素数 *nmemb* の配列を検索して、*key* が指すオブジェクトと一致する要素を見つけます。配列の各要素のサイズは、*size* によって指定されます。

compar が指す比較関数は、*key* オブジェクトと配列要素を順番に指す 2 つの引数を使用して呼び出されます。*key* オブジェクトが配列要素未満、配列要素と等しい、または配列要素より大きいと見なされる場合、それぞれ、ゼロ未満、ゼロと同じ、またはゼロより大きい整数を返さなければなりません。配列は、比較結果が *key* オブジェクト未満であるすべての要素、*key* オブジェクトと等しいすべての要素、*key* オブジェクトより大きいすべての要素という順番で構成されなければなりません。

例

```
/* a sample program that decides if the character
   is in the alphabet */
#include <stdio.h>
#include <stdlib.h>

int main() {
    char *alpha="abcdefghijklmnopqrstuvwxyz";
    char ch, *p;
    int comp();

    ch='m';
```

```
    p=(char *) bsearch(&ch,alpha,26,1,comp);
    if(p)
        printf("%c is in alphabet\n", ch);
    else
        printf("%c is not in alphabet\n", ch);
}

/* compare two characters */
int comp(char *ch, char *s)
{
    return (*ch)-(*s);
}
```

出力

```
m is in alphabet
```

関連項目

calloc

構文

#include <**stdlib.h**>
void * calloc(size_t *nmemb***, size_t** *size***);**

目的

配列に対して領域を割り当てます。

戻り値

calloc() 関数は null ポインタまたは割り当てられた領域へのポインタのいずれかを返します。

パラメータ

nmemb 配列の要素数を示す符号なし整数。
size 配列の各要素のバイト幅を示す整数。

説明

calloc() 関数は、*nmemb* オブジェクトの配列に領域を割り当てます。各領域のサイズは *size* で指定します。領域内のすべてのビットはゼロに初期化されます。

例

```
/* a sample program that allocate 100 floats in memory */
#include <stdio.h>
#include <malloc.h>

int main() {
    float *p;

    p = (float *) calloc(100,sizeof(float));
    if(!p){
      printf("allocation failure - aborting\n");
      exit(1);
    }
    printf ("%f\n", p[8]);
    p[8] = 1.2;
    printf ("%f\n", p[8]);
    free(p);
}
```

出力

```
0.000000
1.200000
```

関連項目

div

構文

#include <**stdlib.h**>
div_t div(int *numer***, int** *denom***);**
ldiv_t div(long int *numer***, long int** *denom***);**

目的

商および剰余を計算します。

戻り値

div() および **ldiv**() 関数はそれぞれ、 **div_t** および **ldiv_t** 型の構造体を返し、商と剰余を構成します。構造体は必ず (順不同の) メンバ *quot*(商) と *rem*(剰余) を含み、それぞれが引数 *numer* および *denom* と同じ型を持ちます。結果に表現できない部分がある場合、動作は不定です。

パラメータ

numer 非除数の整数。
denom 除数の整数。

説明

div() および **ldiv**() 関数は分母 *denom* で分子 *numer* を除算し、1 回の演算で商と剰余を求めます。

例

```
/* a sample program that computes the quotient
and remainder of the division of the numerator
by the denominator. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    int number=400;
    int denom=5;
    div_t result;

    result=div(number,denom);
    printf("%d is divided by %d\n",number,denom);
    printf("the quotient is %d\n",result.quot);
    printf("the remainder is %d\n",result.rem);
}
```

出力

```
400 is divided by 5
the quotient is 80
the remainder is 0
```

関連項目

exit

構文
#include <**stdlib.h**>
void exit(int *status***);**

目的
プログラムを正常に終了します。

戻り値
関数 **exit()** は呼び出し元に戻りません。

パラメータ
status 整数の引数。

説明
exit() 関数はプログラムの正常終了を実行します。1 つのプログラムから **exit()** 関数への複数の呼び出しが実行される場合、動作は不定です。

まず、 **atexit()** 関数によって登録されたすべての関数が、登録順とは逆の順序で呼び出されます。

次に、開いているストリームのうち書き込まれていないバッファデータがあるすべてのストリームがフラッシュされます。その後、開いているすべてのストリームが閉じ、 **tmpfile()** 関数によって作成されたすべてのファイルが削除されます。

最後に、ホスト環境に制御が返されます。*status* の値がゼロまたは **EXIT_SUCCESS** である場合、状態 *successful termination* の実装で定義された形式が返されます。*status* の値が **EXIT_FAILURE** である場合、状態 *unsuccessful termination* の実装で定義された形式が返されます。それ以外の場合、返される状態は実装で定義されます。

例
```
/* a sample program that use exit()*/
#include <stdlib.h>
#include <stdio.h>

int main() {
    printf("exit() is called\n");
    exit(EXIT_SUCCESS);
    printf("This is not called\n");
}
```
出力
```
exit() is called
```

関連項目

free

構文

#include <**stdlib.h**>
void free(void **ptr**);**

目的

領域の割り当てを解除します。

戻り値

free() 関数は値を戻しません。

パラメータ

ptr Void 型の引数へのポインタ。

説明

free() 関数は、*ptr* が指す領域の割り当てを解除します。つまり、その後の割り当てに使用できるようにします。*ptr* が null ポインタである場合は、動作は実行されません。それ以外の場合は、引数が **calloc**() 、 **malloc**() 、または **realloc**() 関数によって以前に返されたポインタと一致しない場合、または、 **free**() または **realloc**() への呼び出しによって領域の割り当てが解除された場合、動作は不定です。

例

```
/* a sample program that allocate 100 floats in memory */
#include <stdio.h>
#include <malloc.h>

int main() {
    float *p;

    p = (float *) calloc(100,sizeof(float));
    if(!p){
        printf("allocation failure - aborting\n");
        exit(1);
    }
    printf("Will now free 100 floats from memory.\n");
    free(p);
}
```

出力

```
Will now free 100 floats from memory.
```

関連項目

getenv

構文

#include <**stdlib.h**>
char * getenv(const char ******name***);**

目的

指定された環境変数の文字列を検索します。

戻り値

getenv() 関数は、指定された文字列と一致した環境変数メンバと関連付けられている文字列へのポインタを返します。ポイントされた文字列をプログラムで変更することはできませんが、 **getenv**() 関数への以降の呼び出しで上書きすることはできます。指定した **name** が見つからない場合は、null ポインタが返されます。

パラメータ

name 環境変数名を表す文字列へのポインタ。

説明

getenv() 関数は、ホスト環境によって提供される環境変数の名前あ g *name* が指す文字列に一致する文字列を検索します。環境名のセットおよび環境リストを変更する方法は、実装で定義されます。

実装では、ライブラリ関数による **getenv**() 関数の呼び出しが存在しないかのように動作する必要があります。

例

出力

関連項目

iscnum

構文

#include <**stdlib.h**>
int iscnum(string_t *pattern*);

目的

文字列が複素数または実数を表すかどうかを調べます。

戻り値

この関数は、文字列が有効な複素数または実数を表す場合は 1 を返し、そうでない場合は 0 を返します。

パラメータ

pattern 入力文字列。

説明

この関数は入力文字列 *pattern* が有効な複素数、純虚数、または実数を表すかどうかを特定します。複素数は complex(re, im)、または、a、$a + ib$、$a + i * b$、$ib + a$、$i * b$ などの従来の数学的表記として表すことができます。ここで、虚数 i は j または I に置換できます。また、a および b は定数であり、正の符号'+' は負の符号'-' に置換できます。

例

```
#include <complex.h>
#include <stdlib.h>
int main() {
   string_t s="3+I*4";

   printf("isnum(\"0x30\") = %d\n", isnum("0x30"));
   printf("isnum(\"abc\") = %d\n", isnum("abc"));
   printf("iswnum(L\"0x30\") = %d\n", iswnum(L"0x30"));
   printf("iswnum(L\"abc\") = %d\n", iswnum(L"abc"));

   printf("iscnum(\"10+I*4\") = %d\n",iscnum(s));
   printf("atoc(s) = %f\n",atoc(s));
   printf("iscnum(\"I*4\") = %d\n",iscnum("I*4"));
   printf("atoc(\"I*4\") = %f\n",atoc("I*4"));
   printf("iscnum(\"complex(1,2)\") = %d\n",iscnum("complex(1,2)"));
   printf("atoc(\"complex(1,2)\") = %f\n",atoc("complex(1,2)"));
   printf("iscnum(\"4\") = %d\n",iscnum("4"));
}
```

出力

```
isnum("0x30") = 1
isnum("abc") = 0
iswnum(L"0x30") = 1
```

```
iswnum(L"abc") = 0
iscnum("10+I*4") = 1
atoc(s) = complex(3.000000,4.000000)
iscnum("I*4") = 1
atoc("I*4") = complex(0.000000,4.000000)
iscnum("complex(1,2)") = 1
atoc("complex(1,2)") = complex(1.000000,2.000000)
iscnum("4") = 0
```

関連項目

isnum(), **iswnum**(), **isenv**(), **iskey**(), **isvar**(), **isstudent**().

---

isenv

構文

#include <**stdlib.h**>
int isenv(**string_t** *name*);

目的

文字列が環境変数名であるかどうかを調べます。

戻り値

この関数は、文字列が環境変数名である場合は 1 を返し、そうでない場合は 0 を返します。

パラメータ

name 入力文字列。

説明

この関数は入力文字列 *name* が環境変数名であるかどうかを特定します。

例

```
/* test if a string is an environment variable */
#include <stdlib.h>
#include <string.h>

int main() {
  int i;

  printf("isenv(\"CHHOME\") = %d\n", isenv("CHHOME"));
  printf("isenv(\"unknown\") = %d\n", isenv("unknown"));
}
```

出力

```
isenv("CHHOME") = 1
isenv("unknown") = 0
```

関連項目

iskey(), **isnum**(), **isstudent**(), **isvar**().

isnum

構文
#include <**stdlib.h**>
int isnum(**string_t** *pattern*);

目的

文字列が数値を表すかどうかを調べます。

戻り値
この関数は、文字列が有効な数値を表す場合は 1 を返し、そうでない場合は 0 を返します。

パラメータ
pattern 入力文字列。

説明
この関数は入力文字列 *pattern* が有効な数値を表すかどうかを特定します。

例
isnum() を参照してください。

関連項目
iscnum(), **iswnum**(), **isenv**(), **iskey**(), **isvar**(), **isstudent**().

iswnum

構文
#include <**stdlib.h**>
int iswnum(**wchar_t** *pattern*);

目的
ワイド文字列が数値を表すかどうかを調べます。

戻り値
この関数は、文字列が有効な数値を表す場合は 1 を返し、そうでない場合は 0 を返します。

パラメータ
pattern 入力文字列。

説明
この関数は入力文字列 *pattern* が有効な数値を表すかどうかを特定します。

例
iscnum() を参照してください。

関連項目
isnum(), **iscnum**(), **isenv**(), **iskey**(), **isvar**(), **isstudent**().

malloc

構文

#include <**stdlib.h**>
void * malloc(size_t*size***);**

目的

オブジェクトに対して領域を割り当てます。

戻り値

malloc() 関数は、null ポインタまたは割り当てられた領域に対するポインタのいずれかを返します。

パラメータ

size 割り当てる領域の大きさを表す整数。

説明

malloc() 関数は、*size* によってサイズが指定され、値が不定であるオブジェクトに対して領域を割り当てます。

例

```
/* a sample program that allocates space for a string
   dynamically, request user input, and then print the
   string backwards.  This is executed with the following
   statement:  malloc.c < malloc.in.  The contents of
   malloc.in are: abcd */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main() {
    char *s;
    int t;

    s = malloc(80);
    printf("Please enter a string: ");
    if(!s) {
      printf("Memory request failed.\n");
      exit(1);
    }
    fgets(s, 80, stdin);
    printf("%s", s);
    printf("Converted string: ");
    for(t = strlen(s)-2; t>=0; t--)
      putchar(s[t]);
    printf("\n");
    free(s);
    return (0);
}
```

出力

```
Please enter a string: abcd
Converted string: dcba
```

関連項目

mblen

構文

#include <**stdlib.h**>
int mblen(const char **s***,* **size_t** *n***);**

目的

マルチバイト文字のバイト数を特定します。

戻り値

s が null ポインタである場合、 **mblen**() 関数は、マルチバイト文字エンコーディングに状況依存のエンコーディングがある場合は非ゼロを、ない場合はゼロを返します。*s* が null ポインタでない場合、**mblen**() 関数はゼロを返すか (*s* が null 文字を指している場合)、マルチバイト文字に含まれるバイト数を返すか (次の *n* バイトまたはそれより少ないバイトで有効なマルチバイト文字を形成する場合)、-1 を返します (有効なマルチバイト文字を形成しない場合)。

パラメータ

s テストされる文字へのポインタ。
n テストされるバイト数を指定する符号なし整数の引数。

説明

s が null ポインタでない場合、 **mblen**() 関数は、*s* が指すマルチバイト文字に含まれているバイト数を特定します。 **mbtowc** 関数のシフト状態に影響がないこと以外は、次と同じです。

mbtowc((wchar_t *)0, s, n);

実装では、ライブラリ関数による **mblen**() 関数の呼び出しが存在しないかのように動作する必要があります。

例

```
/* a sample program that displays the length of the
multibyte character pointed by mb. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    char mb[] = "This is a test";

    printf("The length of the multibyte character pointed by str is %d\n" \
    ,mblen(mb,sizeof(char)));
}
```

出力

```
The length of the multibyte character pointed by str is 1
```

関連項目

mbstowcs

構文

#include <**stdlib.h**>
size_t mbstowcs(wchar_t * restrict *pwcs***, const char restrict** **s***, size_t** *n***);**

目的

マルチバイト文字のシーケンスを対応するコードに変換します。

戻り値

無効なマルチバイト文字が検出されると、 **mbstowcs()** 関数は (**size_t**) – **1** を返します。それ以外の場合は、変更された配列要素数を返します。終了ゼロコードがあっても、これには含まれません。

パラメータ

pwcs 変換後のワイド文字列を格納する配列へのポインタ。
s 変換されるマルチバイト文字列へのポインタ。
n 変換されるマルチバイト文字数。

説明

mbstowcs() 関数は、*s* が指す配列から初期シフト状態で始まるマルチバイト文字のシーケンスを対応するコードのシーケンスに変換し、*pwcs* が指す配列に最大 *n* コードを格納します。null 文字に続くマルチバイト文字 (値ゼロを保持するコードに変換される) は、検査または変換されません。各マルチバイト文字は、mbtowc() 関数のシフト状態に影響がないこと以外は、 **mbtowc()** 関数への呼び出しと同様に変換されます。
pwcs が指す配列では、*n* 個以下の要素が変更されます。重複するオブジェクト間でコピーが実行される場合、動作は不定です。

例

```
/* a sample program that converts the first 4 characters
in the multibyte character pointed by mb and puts the result
in str. */
#include <wchar.h>
#include <stdlib.h>

int main() {
    char s[] = "This is a test";
    wchar_t pwcs[20];
    size_t n;

    n = strlen(s)+1;
    printf("Original string: %s\n", s);
    printf("Number of characters to convert: %d\n", n);
    if(mbstowcs(pwcs,s,n) == -1) {
        printf("error convert\n");
        exit(1);
```

```
    }
    printf("The converted characters are: ");
    fputws(pwcs, stdout);
    fputs("\n", stdout);
}
```

出力

```
Original string: This is a test
Number of characters to convert: 15
The converted characters are: This is a test
```

関連項目

mbtowc

関連項目

#include <**stdlib.h**>
int mbtowc(wchar_t * restrict *pwc***, const char restrict** **s***, size_t** *n***);**

目的

マルチバイト文字をワイド文字に変換します。

戻り値

s が null ポインタである場合、 **mbtowc**() 関数は、マルチバイト文字エンコーディングに状況依存のエンコーディングがある場合は非ゼロを、ない場合はゼロを返します。*s* が null ポインタでない場合、**mbtowc**() 関数はゼロを返すか (*s* が null 文字を指している場合)、変換後のマルチバイト文字に含まれるバイト数を返すか (次の *n* バイトまたはそれより少ないバイトで有効なマルチバイト文字を形成する場合)、-1 を返します (有効なマルチバイト文字を形成しない場合)。

いずれの場合にも、*n* より大きい値または **MB_CUR_MAX** マクロの値は返されません。

パラメータ

pwc 変換後のワイド文字列を格納する配列へのポインタ。
s 変換されるマルチバイト文字列へのポインタ。
n 変換されるマルチバイト文字。

説明

s が null ポインタでない場合、 **mbtowc**() 関数は、*s* が指すマルチバイト文字に含まれているバイト数を特定します。次に、マルチバイト文字に対応する **wchar_t** 型の値のコードを特定します。(null 文字に対応するコードの値はゼロです。) マルチバイト文字が有効であり、*pwc* が null ポインタでない場合、 **mbtowc**() 関数は *pwc* が指すオブジェクトにコードを格納します。*s* が指す配列の最大 *n* バイトが検査されます。
実装では、ライブラリ関数による **mbtowc**() 関数の呼び出しが存在しないかのように動作する必要があります。

例

```
/* a sample program that converts the multibyte character in s
into its equivalent wide character and puts the result in the
array  pointed to by pwcs. */
#include <wchar.h>
#include <stdlib.h>

int main() {
    char s[] = "This is a test";
    wchar_t pwcs[8];
    size_t n;
```

```
    n = 1;
    if(mbtowc(pwcs,s,n) == -1) {
      printf("error convert\n");
      exit(1);
    }
    printf("The original string is: %s\n", s);
    printf("The converted characters are: ");
    fputws(pwcs, stdout);
    fputs("\n", stdout);
}
```

出力

```
The original string is: This is a test
The converted characters are: T
```

関連項目

qsort

構文

#include <**stdlib.h**>
void qsort(const void **base***,* **size_t** *nmemb***,* **size_t** *size***,* **int (****compar*)**(const void *, const void *));**

目的

配列を並べ替えます。

戻り値

qsort() 関数は値を返しません。

パラメータ

base ソートする配列へのポインタ。
nmemb 配列の要素数。
size 配列要素のバイト幅。
compar 比較関数へのポインタ。

説明

qsort() 関数は *nmemb* 個の要素からなる配列を並び替えます。最初の要素は *base* によって参照されます。各要素のサイズは、*size* によって指定されます。

配列の内容は *compar* が指す比較関数に従って昇順に並び替えられます。関数は、比較対象のオブジェクトを指す 2 つの引数を使用して呼び出されます。関数は、最初の引数が 2 番目の引数未満、2 番目の引数と等しい、または 2 番目の引数より大きいと見なされる場合、それぞれ、ゼロ未満、ゼロと同じ、またはゼロより大きい整数を返さなければなりません。

例

```
/* a sample program that sorts a list of integers and
displays the result. */
#include <stdio.h>
#include <stdlib.h>

int num[10] = { 1,3,6,5,8,7,9,6,2,0};
int comp(const void *,const void *);

int main() {
    int i;

    printf("Original array:\t");
    for(i=0; i<10; i++)
        printf("%d ",num[i]);
    qsort(num,10,sizeof(int),comp);
    printf("\nSorted array:\t");
    for(i=0; i<10; i++)
```

```
        printf("%d ",num[i]);
    printf("\n");
}

/* compare the integers */
int comp(const void *i, const void *j) {
    return *(int *)i - *(int *)j;
}
```

出力

```
Original array: 1 3 6 5 8 7 9 6 2 0
Sorted array: 0 1 2 3 5 6 7 6 8 9
```

関連項目

rand

構文

#include <**stdlib.h**>
int rand(*void***);**

目的

擬似乱数整数を返します。

戻り値

rand() 関数は擬似乱数整数を返します。

パラメータ

引数はありません。

説明

rand() 関数は、0 から **RAND_MAX** の範囲で擬似乱数整数のシーケンスを計算します。

例

```
/* a sample program that displays ten pseudorandom numbers. */
#include <stdio.h>
#include <stdlib.h>

int main() {
    int i;

    for(i=0; i<10; i++) printf("%d ",rand());
    printf ("\n");
}
```

出力

```
16838 5758 10113 17515 31051 5627 23010 7419 16212 4086
```

関連項目

realloc

構文

#include <**stdlib.h**>
void * realloc(void ******ptr***, size_t** *size***);**

目的

メモリブロックの再割り当てを行ないます。

戻り値

realloc() 関数は、新しいオブジェクト (古いオブジェクトへのポインタと同じ値を持つかも知れない) へのポインタを返すか、新しいオブジェクトを割り当てることができない場合は、null ポインタを返します。

パラメータ

ptr 割り当て済みメモリブロックへのポインタ。
size 新しいサイズ (バイト単位)。

説明

realloc() 関数は、*ptr* が指す古いオブジェクトの割り当てを解除し、*size* で指定されるサイズを持つ新規のオブジェクトへのポインタを返します。新規オブジェクトの内容は割り当て解除前の古いオブジェクトと一致し、サイズは新しいサイズと古いサイズのうち小さい方に一致します。古いオブジェクトのサイズを超える新規オブジェクトのバイト部分の値は不定です。

ptr が null ポインタである場合、 **realloc**() 関数は指定されたサイズの **malloc**() 関数と同様に動作します。それ以外の場合は、*ptr* が **calloc**() 、**malloc**() 、または **realloc**() 関数によって以前に返されたポインタと一致しない場合、または、 **free**() 関数または **realloc**() 関数への呼び出しによって領域の割り当てが解除された場合、動作は不定です。新規オブジェクトにメモリを割り当てることができない場合は、古いオブジェクトへの割り当てが解除されず、その値は変わりません。

例

```
/* a sample program that first allocates 16 characters,
copies the string "This is 16 chars" into them, and then
use realloc() to increase the size to 17 in order to place
a period at the end. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char *p;

    p = malloc(16);
    if(!p) {
```

```
        printf("Allocation error - aborting.");
        exit(1);
    }
    strcpy(p,"This is 16 chars");
    p = realloc(p,17);
    if(!p) {
        printf("allocation error - aborting.");
        exit(1);
    }
    strcat(p,".");
    printf(p);
    printf("\n");
    free(p);
}
```

出力

```
This is 16 chars.
```

関連項目

remenv

構文

#include <**stdlib.h**>
int remenv(**string_t** *name*);

目的

環境変数を削除します。

戻り値

関数 remenv() は、成功すると 0 を返し、失敗すると-1 を返します。

パラメータ

name 削除される環境変数名を含む文字列。

説明

この関数は、文字列 *name* 内の名前を持つ環境変数を削除します。

例

```
#include <stdio.h>
#include <stdlib.h>

int main() {
  string_t s;
  putenv("TESINGENV=testingenv");
  s = getenv("TESINGENV");
  printf("s = %s\n", s);

  remenv("TESINGENV");
  s = getenv("TESINGENV");
  if(s == NULL)
    printf("ok: s = NULL\n");
  else
    printf("error: s = %s\n", s);
}
```

出力

```
s = testingenv
ok: s = NULL
```

関連項目

putenv().

srand

構文

#include <**stdlib.h**>
void srand(unsigned int *seed***);**

目的

擬似乱数の初期値を設定します。

戻り値

srand() 関数は値を返しません。

パラメータ

seed 符号なし整数の引数。

説明

srand() 関数では、引数を後続の **rand()** への呼び出しによって返される擬似乱数の新しいシーケンスのシードとして使用します。次に、 **srand()** が同じシード値を使用して呼び出される場合、その擬似乱数のシーケンスが必ず繰り返されます。 **srand()** の呼び出しを一度も行わないまま **rand()** を呼び出した場合、シード値 1 を使用した **srand()** の最初の呼び出しと同じシーケンスが必ず生成されます。

実装では、ライブラリ関数による **srand()** 関数の呼び出しが存在しないかのように動作する必要があります。

例

```
/* a sample program that displays ten pseudorandom numbers. */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    int i,stime;
    long ltime;

    ltime = time(NULL); /* get current calendar time */
    stime = (unsigned int) ltime/2;
    srand(stime);
    for(i=0; i<10; i++) printf("%d ",rand());
    printf("\n");
}
```

出力

```
17875 24317 11838 16984 8536 4774 910 5469 11847 28683
```

関連項目

strtod

構文

#include <**stdlib.h**>
double strtod(const char * *restrict nptr***, char** ** *restrict endptr***);**

目的

文字列の一部を double 型の表現に変換します。

戻り値

変換された値がある場合は、その値を返します。変換が実行できなかった場合は、ゼロが返されます。正確な値が表現可能な値の範囲外である場合、正または負の **HUGE_VAL**、 **HUGE_VALF**、または **HUGE_VALL** が返され (戻り値の型と値の符号に従う)、マクロ **ERANGE** の値は **errno** に格納されます。結果がアンダーフローする場合、絶対値が戻り値の型の正規化された正の数の最小値以下である値を返します。 **errno**() が値 **ERANGE** を取得するかどうかは実装で定義されます。

パラメータ

nptr double 型数値に変換する文字列へのポインタ。
endptr 走査を終了した文字へのポインタ。

説明

strtod() 関数は、 **nptr** が指す文字列の最初の部分を **double** 型の表現に変換します。まず、入力文字列を 3 つの部分に分解します。最初の部分は、空白文字のシーケンス (**isspace**() 関数の指定に準じる) となり、空の場合もあります。次の部分は、浮動小数点の定数に類似のシーケンスか、無限大または NaN を表す対象シーケンスです。最後の部分は、入力文字列の終端の null 文字を含む 1 つ以上の認識されない文字列です。次に、対象シーケンスを浮動小数点数に変換し、その結果を返します。

対象シーケンスの期待される形式は、オプションの正または負符号を持つ次のいずれかです。

— 10 進数字の空でないシーケンス (オプションで小数点文字を含む) と、オプションの指数部分。

— **0x** または **0X** の後に、16 進数字の空でないシーケンス (オプションで小数点文字を含む) と、オプションの 2 進数の指数部分。

— **inf** または **infinity** のいずれか。大文字と小文字を区別しません。

— **NAN** または **NAN(***n-char-sequence*$_{opt}$**)** のいずれか。 **NAN** 部分の大文字と小文字を区別しません。

次のようになります。

n-char-sequence
digit
nondigit
n-char-sequence digit
n-char-sequence nondigit

対象シーケンスは、入力文字列の最長の初期サブシーケンスとして定義されます。このシーケンスは、先頭が空白文字以外で始まり、期待される形式をとります。入力文字列が期待される形式でない場合、対象シーケンスに文字は含まれません。

対象シーケンスが浮動小数点数の期待される形式を持つ場合、1 桁目の数字または小数点文字 (いずれか最初に検出される方) で始まる文字のシーケンスは浮動小数点の定数として解釈されます。このとき、ピリオドの代わりに使用される小数点文字は除外されます。また、10 進数浮動小数点数に指数部分または小数点文字のいずれも出現しないか、2 進数浮動小数点に 2 進数指数部分が出現しない場合は、値ゼロを持つ適切な型の指数部分が文字列の最終桁の後に続くと想定されます。対象シーケンスの先頭が負符号である場合、そのシーケンスは負と解釈されます。戻り値の型で表現できる場合、文字シーケンス **INF** または **INFINITY** は無限大として解釈されます。その他の場合は、戻り値の型の範囲より大きい浮動小数点定数などです。文字シーケンス **NAN** または **NAN (***n-char-sequence*$_{opt}$**)** が戻り値の型でサポートされる場合、シグナルなしの NaN として解釈されます。その他、期待される形式を持たない対象シーケンスなどの場合、n-char シーケンスの意味は実装で定義されます。最後の文字列へのポインタは、**endptr** が指すオブジェクトに格納されます。ただし、**endptr** が null ポインタでないことが条件です。

対象シーケンスが 16 進数形式であり、 **FLT_RADIX** が 2 の累乗である場合、変換結果の値は正しく丸められます。

"C" ロケール以外のロケールが指定されている場合には、追加のロケール固有の対象シーケンス形式も使用できます。

対象シーケンスが空であり、期待される形式でない場合、変換は実行されません。 **nptr** の値は **endptr** が指すオブジェクトに格納されます。ただし、**endptr** が null ポインタでないことが条件です。

推奨事項
対象シーケンスが 16 進数の形式であり、 **FLT_RADIX** が 2 の累乗でない場合、結果は 16 進浮動小数点のソース値に隣接する 2 つの数字のいずれかになり、適切な内部形式で表されます。ただし、誤差を表す符号は、現在の丸め方向に一致します。

対象シーケンスが 10 進数形式であり、**float.h** で定義される **DECIMAL_DIG** 以下の有効桁数を持つ場合、結果は正しく丸められます。対象シーケンス *D* が 10 進数形式であり、 **DECIMAL_DIG** の有

効桁数を超える場合は、2 つの隣接する 10 進数文字列 L および U を検討します。これらは両方とも **DECIMAL_DIG** の有効桁数を持ち、L、D、および U の値は $L \leq D \leq U$ を満たします。結果は常に、現在の丸め方向に従って L および U を正しく丸めることによって取得される等しい値または隣接する値の 1 つになります。ただし、追加条件として、D に関する誤差を表す符号は現在の丸め方向に一致します。

例

出力

関連項目

strtol

構文

#include <**stdlib.h**>
long int strtol(const char * *restrict nptr***, char** ** *restrict endptr***, int** *base***);**
long long int strtoll(const char * *restrict nptr***, char** ** *restrict endptr***, int** *base***);**
unsigned long int strtoul(const char * *restrict nptr***, char** ** *restrict endptr***, int base);**

目的

文字列の一部を **long int**、**long long int**、および **unsigned long int** 型の表現に変換します。

戻り値

strtol() 、 **strtoll**() 、および **strtoul**() 関数は、変換された値がある場合にその値を返します。変換が実行できなかった場合は、ゼロが返されます。正確な値が表現可能な値の範囲外である場合、(戻り値の型と値の符号に応じて) **LONG_MIN**、 **LONG_MAX**、 **LLONG_MIN**、 **LLONG_MAX**、または **ULONG_MAX** が返されます。また、マクロ **ERANGE** の値が **errno** に格納されます。

パラメータ

nptr 整数型に変換する文字列へのポインタ。
endptr 走査を終了した文字へのポインタ。
base 基数を表す整数。

説明

strtol() 、 **strtoll**() 、および **strtoul**() 関数は、*nptr* が指す文字列の最初の部分を **long int**、**long long int**、および **unsigned long int** 型の表現にそれぞれ変換します。まず、入力文字列を 3 つの部分に分解します。最初の部分は、空白文字のシーケンス (**isspace**() 関数の指定に準じる) となり、空の場合もあります。次の部分は、*base* の値によって特定される基数で表現される整数に類似した対象シーケンスです。最後の部分は、入力文字列の終端の null 文字を含む 1 つ以上の認識されない文字列です。次に、対象のシーケンスを整数に変換し、その結果を返します。

base の値がゼロである場合、対象シーケンスの期待される形式は、整数定数形式です。オプションで先頭に正または負符号が付きますが、整数のサフィックスは付きません。*base* の値が 2 から 36 の間 (両端の数値を含む) である場合、対象シーケンスの期待される形式は、*base* で指定された基数を使用して整数を表現する文字および数字のシーケンスです。オプションで先頭に正または負符号が付きますが、整数のサフィックスは付きません。**a** (または **A**)〜**z** (または **Z**) の文字は 10〜35 の値に属します。属する値が文字および数字のみの場合は、*base* より小さい値が許可されます。*base* の値が 16 である場合、オプションで文字 **0x** または **0X** を文字および数字のシーケンスの前に配置し、符号があればその後に続けます。

対象シーケンスは、入力文字列の最長の初期シーケンスとして定義されます。このシーケンスは、先頭が空白文字以外で始まり、期待される形式をとります。入力文字列が空の場合か全体が空白で構成されている場合、または、最初の空白以外の文字が符号以外であるか許可できる文字または数字である場合、対象シーケンスに文字は含まれません。

対象シーケンスが期待される形式であり、*base* の値がゼロである場合、1 桁目の数字で始まる文字のシーケンスは整数の定数として解釈されます。対象シーケンスが期待される形式であり、*base* の値が 2〜36 である場合、その値が変換の基点として使用され、各文字に上述の値が割り当てられます。対象シーケンスの先頭が負の符号である場合、変換の結果の値は負と解釈されます (戻り値の型)。最後の文字列へのポインタは、*endptr* が指すオブジェクトに格納されます。ただし、*endptr* が null ポインタでないことが条件です。
“C” ロケール以外に追加のロケール固有の対象シーケンス形式も使用できます。

対象シーケンスが空であり、期待される形式でない場合、変換は実行されません。*nptr* の値は *endptr* が指すオブジェクトに格納されます。ただし、*endptr* が null ポインタでないことが条件です。

例
出力
関連項目

system

構文

#include <**stdlib.h**>
int system(const char ******string***);**

目的

コマンドプロセッサを使用してプログラムを実行します。

戻り値

引数が null ポインタである場合、 **system()** 関数はコマンドプロセッサが使用可能である場合に限り非ゼロを返します。引数が null ポインタでない場合は、 **system()** 関数から戻るときに、実装で定義されている値を返します。

パラメータ

string コマンドプロセッサに渡される文字列へのポインタ。

説明

string が null ポインタである場合、 **system()** 関数はホスト環境にコマンドプロセッサが備わっているかどうかを特定します。*string* が null ポインタでない場合、 **system()** 関数は *string* が指す文字列をそのコマンドプロセッサに渡します。コマンドプロセッサの実行方法は実装によって文書化されなければならないことになっています。このため、 **system()** を呼び出すプログラムが非準拠の方法で動作したり中止されたりすることがあります。

例

出力

関連項目

wcstombs

構文

#include <**stdlib.h**>
size_t wcstombs(char restrict **s***,* **const wchar_t * restrict** *pwcs*, **size_t** *n*);

目的

コードのシーケンスを対応するマルチバイト文字に変換します。

戻り値

有効なマルチバイト文字と対応しないコードが検出されると、 **wcstombs**() 関数は (**size_t**) – **1** を返します。それ以外の場合、 **wcstombs**() 関数は変更後のバイト数を返します。終端の null 文字があっても、これには含まれません。

パラメータ

pwcs 変換されるワイド文字列へのポインタ。
s 変換後のマルチバイト文字列を格納する配列へのポインタ。
n 変換後のマルチバイト文字の最大文字数。

説明

wcstombs() 関数は、*pwcs* が指す配列のマルチバイト文字に対応するコードのシーケンスを、初期シフト状態で始まるマルチバイト文字のシーケンスに変換して、これらのマルチバイト文字を *s* が指す配列に格納します。マルチバイト文字が制限 *n* を超える場合、または、null 文字が格納される場合は停止します。各コードは、 **wctomb**() 関数のシフト状態に影響がないこと以外は、 **wctomb**() 関数への呼び出しによる場合と同様に変換されます。

pwcs が指す配列では、*n* 個以下の要素が変更されます。重複するオブジェクト間でコピーが実行される場合、動作は不定です。

例

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    char s[8];
    wchar_t pwcs[8] = L"123abc\n";
    size_t retval;

    retval = wcstombs(s,pwcs, wcslen(pwcs)+1);
    if(retval != (size_t) -1) {
        printf("the original string is: ");
        fputws(pwcs, stdout);
     printf("The converted result is: %s\n",s);
    }
```

```
    else
printf("Error\n");
}
```

出力

```
the original string is: 123abc
The converted result is: 123abc
```

関連項目

wctomb

構文

#include <**stdlib.h**>
int wctomb(char **s***, **wchar_t** *wchar***);**

目的

ワイド文字をマルチバイト文字に変換します。

戻り値

s が null ポインタである場合、 **wctomb**() 関数は、マルチバイト文字エンコーディングに状況依存のエンコーディングがある場合は非ゼロを、ない場合はゼロを返します。**s** が null ポインタでない場合、*wchar* の値が有効なマルチバイト文字に一致しない場合は-1 を返し、それ以外の場合は *wchar* の値に対応するマルチバイト文字が含むバイト数を返します。
いずれの場合にも、 **MB_CUR_MAX** マクロの値より大きい値は返されません。

パラメータ

s 変換後のマルチバイト文字列へのポインタ。
wchar 変換されるワイド文字列へのポインタ。

説明

wctomb() 関数は、値が *wchar* であるコード (シフト状態の変更を含む) に対応するマルチバイト文字を表すのに必要なバイト数を特定します。この関数では、マルチバイト文字表現を *s* が指す配列オブジェクトに格納します (*s* が null ポインタでない場合)。最大 **MB_CUR_MAX** 文字が格納されます。*wchar* の値がゼロである場合、wctomb 関数の初期シフト状態のままになります。

実装では、ライブラリ関数による **wctomb**() 関数の呼び出しが存在しないかのように動作する必要があります。

例

```
/* This is an example of wctomb(char*, wchar_t).  This function converts
   a wide character to the corresponding multibyte character.
*/
#include <stdio.h>
#include <stdlib.h>

int main() {
    char s[128];
    wchar_t pwcs;
    int retval,x;
    pwcs = 'T';
    wctomb(s,pwcs);
    if(retval != -1) {
      printf("The converted result is: %s\n",s);
```

```
    }
    else
printf("Error\n");
}
```

出力

```
The converted result is: T
```

関連項目

第19章　文字列関数 — <string.h>

ヘッダー **string.h** には、1 つの型と複数の関数が宣言されています。宣言されている型は **size_t** です。配列の長さはさまざまな方法で決定されますが、すべての場合において **char *** または **void *** 引数は配列の先頭 (最下位アドレス) の文字を指します。配列がオブジェクトの終わりを越えてアクセスされた場合、動作は不定です。

size_t n として宣言された引数が関数の配列の長さを指定する場合、**n** はその関数への呼び出しでゼロの値を持つことがあります。この副分節の特定の関数の説明で明記しない限り、このような呼び出しのポインタ引数も有効な値を持つものとします。このような呼び出しでは、文字を検索する関数で文字の出現箇所が見つからない場合、文字をコピーする関数はゼロ個の文字をコピーします。

関数
string ヘッダーを使用して実装される関数は次のとおりです。

関数	説明
memchr	ストリーム内で、指定された文字の最初の出現箇所を検索します。
memcmp	2 つのオブジェクト内の最初の文字を比較します。
memcpy	オブジェクトの文字を別のオブジェクトにコピーします。
memmove	オブジェクトの文字を別のオブジェクトにコピーします。
memset	オブジェクトの指定された数の位置に単一の値をコピーします。
str2ascii	各文字の ASCII 値を返します。
str2mat	個々の文字列から char の行列を作成します。
stradd	2 つの文字列を足します。
strcasecmp	すべての文字を小文字に変換した後、2 つの文字列を比較します。
strcat	文字列のコピーを別の文字列に追加します。
strchr	文字列内で文字の最初の出現箇所を検索します。
strcmp	2 つの文字列を比較します。
strcoll	2 つの文字列を比較します。
strconcat	文字列のコピーを文字列に追加します。
strcpy	文字列を配列にコピーします。
strcspn	文字列で、先頭からのセグメントの最大の長さを計算します。
strdup	文字列を複製します。
strerror	**errnum** の値をマップします。
strgetc	指定した場所の文字を返します。

strjoin	複数の文字列を 1 つの文字列として結合します。
strlen	文字列の長さを計算します。
strncasecmp	すべての文字を小文字に変換した後、2 つの文字列を比較します。
strncat	配列の文字をワイド文字列に追加します。
strncmp	2 つの配列間で文字を比較します。
strncpy	配列の文字を別の配列にコピーします。
strpbrk	文字列内の文字の最初の出現箇所を別の文字列内で検索します。
strputc	指定した場所の文字を置き換えます。
strrchr	文字列内での文字の最後の出現箇所を検索します。
strrep	別の文字列内で文字列を検索し、置き換えます。
strspn	文字列で、先頭からのセグメントの最大の長さを計算します。
strstr	文字列内のオブジェクトの最初の出現箇所を別の文字列内で検索します。
strtok	文字列をトークンのシーケンスに分割します。
strtok_r	文字列を再入可能なトークンのシーケンスに分割します。
strxfrm	文字列を変換し、配列内に配置します。

マクロ

string ヘッダーではマクロは定義されていません。

宣言される型

string ヘッダーには次の型が宣言されています。

宣言される型	説明
size_t	符号なし整数。

移植性

C と Ch の相違点

関数 **strcpy**、 **strncpy**、 **strcat**、 **strncat**、 **strcmp**、 **strcoll**、 **strtolc**、 **memcpy**、および **memmove** は、Ch の組み込み関数です。

str2ascii、**str2mat**、**strgetc**、**strputc**、および **strrep** は Ch でのみ使用可能です。

memchr

構文

#include <**string.h**>
void *memchr(const void **s***,* **int** *c***,** **size_t** *n***);**

目的

バッファ内で、指定された文字の最初の出現箇所を検索します。

戻り値

memchr() 関数は、検出した文字へのポインタを返します。または、オブジェクト内に文字が出現しなかった場合は、null ポインタを返します。

パラメータ

s 検索対象となるバッファへのポインタ。
c 検索する文字。
n 検索する文字数。

説明

memchr() 関数は、**s** が指すオブジェクトの先頭から **n** 個の文字 (それぞれが **unsigned char** として解釈される) で、**c** (**unsigned char** に変換される) の最初の出現箇所を検索します。

例

```
/* a sample program that prints 'is a test' on the screen.*/
#include <stdio.h>
#include <string.h>

int main() {
    char p[] = "this is a test\n";
    char *s;

    s = memchr(p,' ',strlen(p));
    printf(s);
}
```

出力

```
 is a test
```

関連項目

memcmp

構文

#include <**string.h**>
int memcmp(const void **s1***,* **const void** **s2***,* **size_t** *n***);**

目的

2 つのオブジェクト内の最初の **n** 文字を比較します。

戻り値

memcmp() 関数は、**s1** が指すオブジェクトが、**s2** が指すオブジェクトよりも大きいか、等しいか、小さいかに応じて、ゼロより大きい整数、ゼロに等しい整数、またはゼロより小さい整数を返します。

パラメータ

s1 比較対象の一方の文字列へのポインタ。
s2 比較対象の他方の文字列へのポインタ。
n 比較する文字数。

説明

memcmp() 関数は、**s1** が指すオブジェクトの最初の **n** 個の文字と、**s2** が指すオブジェクトの最初の **n** 個の文字を比較します。

例

```
/* a sample program that shows the outcome of a comparison
of its two strings.*/
#include <stdio.h>
#include <string.h>

int main() {
    char s1[] = "abcdefghijkl";
    char s2[] = "abcdefkl";
    int outcome;
    int l1,l2;
    int len;

    len = (l1 = strlen(s1)) < (l2 = strlen(s2)) ? l1:l2;
    outcome = memcmp(s1,s2,len);
    if(! outcome) printf("equal\n");
    else if(outcome<0) printf("first less than second\n");
    else printf("first greater than second\n");
}
```

出力

```
first less than second
```

関連項目

memcpy

構文

#include <**string.h**>
void *memcpy(void * restrict *s1***, const void * restrict** *s2***, size_t** *n***);**

目的

オブジェクトの文字を別のオブジェクトにコピーします。

戻り値

memcpy() は値 **s1** を返します。

パラメータ

s1 コピー先バッファへのポインタ。
s2 コピー元文字列へのポインタ。
n コピーする文字数。

説明

memcpy() 関数は、**s2** が指すオブジェクトの **n** 個の文字を **s1** が指すオブジェクトにコピーします。重複するオブジェクト間でコピーが実行された場合、動作は不定です。

例

出力

関連項目

memmove

構文

#include <**string.h**>
void *memmove(void **s1**, **const void** **s2**, **size_t** *n*);

目的

オブジェクトの文字を別のオブジェクトにコピーします。

戻り値

memmove() は値 **s1** を返します。

パラメータ

s1 コピー先バッファへのポインタ。
s2 コピー元文字列へのポインタ。
n コピーする文字数。

説明

memmove() 関数は、**s2** が指すオブジェクトから **s1** が指すオブジェクトに **n** 個の文字をコピーします。コピーの動作は次のように行われます。まず、**s2** が指すオブジェクトから **n** 個の文字が、**n** 個の文字から成り **s1** または **s2** が指すオブジェクトと重ならない一時配列にコピーされます。次に、その一時配列から **s1** が指すオブジェクトに **n** 個の文字がコピーされます。

例

出力

関連項目

memset

構文

#include <**wchar.h**>
void *memset(void **s***,* **int** *c*, **size_t** *n*);

目的

バッファの先頭から連続して、指定された個数だけ単一の値をコピーします。

戻り値

memset() は値 **s** を返します。

パラメータ

s コピー先バッファへのポインタ。
c コピーする文字。
n コピー後の文字 **c** の個数。

説明

memset() 関数は、**c** の値 (**unsigned char** に変換される) を、**s** が指すオブジェクトの先頭から **n** 個の各文字にコピーします。

例

出力

関連項目

str2ascii

構文

#include <**string.h**>
unsigned int str2ascii(string_t *s***);**

目的

文字列の ASCII 数値を取得します。

戻り値

この関数は文字列の各文字の ASCII 値の合計を返します。

パラメータ

s 入力文字列。

説明

関数 **str2ascii()** は文字列の各文字の ASCII 値の合計を返します。

例

```
#include <stdio.h>
#include <string.h>

int main() {
   int ascii;
   ascii = str2ascii("abcd");
   printf("ASCII num for string `abcd` is %d\n", ascii);
}
```

出力

```
ASCII num for string `abcd` is 394
```

関連項目

str2mat

構文

#include <**string.h**>
int str2mat(array char *mat*[:][:], **string_t** *s*, ...);

目的

個々の文字列から char の行列を作成します。

戻り値

この関数は、成功すると 0 を返し、失敗すると-1 を返します。

パラメータ

mat 作成された行列を含む 2 次元配列。
s 個々の文字列。

説明

この関数は、文字列 *s* および可変長の引数から渡される文字列を含む char の行列を作成します。*s* および可変長の引数を含む文字列の総数は、配列 *mat* の最初の次元内の要素数と等しくなければなりません。文字列数が配列 *mat* の要素数より少ない場合、この関数呼び出しは失敗し、-1 を返します。要素数を超える文字列は無視されます。

例

```
#include <stdio.h>
#include <string.h>
#include <array.h>

int main() {
   array char mat[3][10];
   int status;

   mat = (array char [3][10])' '; /* blank space */
   status = str2mat(mat, "abcd");
   printf("status = %d\nmat = %c\n", status, mat);
   str2mat(mat, "abcd", "efghi", "0123456789");
   printf("mat = \n%c\n", mat);
   status = str2mat(mat, "ABCD", "EFGH", "ab23456789", "too many strings");
   printf("status = %d\nmat = \n%c\n", status, mat);
}
```

出力

```
status = 0
mat = a b c d
```

```
mat =
a b c d
e f g h i
0 1 2 3 4 5 6 7 8 9

status = -1
mat =
A B C D
E F G H
a b 2 3 4 5 6 7 8 9
```

関連項目

strputc().

stradd

構文
#include <**string.h**>
string_t stradd(string_t *s***, ...);**

目的
文字列を結合します。

戻り値
stradd() 関数は新しい文字列を返します。

パラメータ
s 結合される文字列。

説明
stradd() 関数は、文字列 **s** および可変長の引数によって渡される後続の文字列を結合します。

例
```
/** add strings */
#include <stdio.h>
#include <string.h>

int main() {
  string_t s, s1 = "ABCD";
  char *s2="cbcd";
  char s3[]="1234";

  s = stradd(s1, s2, s3);
  printf("s = %s\n", s);
  s = stradd(s2, s3);
  printf("s = %s\n", s);
}
```

出力
```
s = ABCDcbcd1234
s = cbcd1234
```

関連項目

strcasecmp

構文

#include <**string.h**>
int strcasecmp(const char **s1***,* **const char** **s2***);*

目的

2 つの文字列を比較します。

戻り値

strcasecmp() 関数は、**s1** が指す文字列が、**s2** が指す文字列より大きいか、等しいか、小さいかに応じて、ゼロより大きい整数、ゼロに等しい整数、またはゼロより小さい整数を返します。

パラメータ

s1 比較対象の一方である文字列へのポインタ。
s2 比較対象の他方である文字列へのポインタ。

説明

strcasecmp() 関数は **s1** が指す文字列と **s2** が指す文字列を比較します。文字列を比較する前に、**tolower()** 関数が呼び出され大文字が小文字に変換されます。

例

```
/* a sample program that shows the outcome of a comparison
of its two strings.*/
#include <stdio.h>
#include <string.h>

int main() {
    char s1[] = "abc";
    char s2[] = "abc";
    int outcome;
    int l1,l2;

    outcome = strcasecmp(s1,s2);
    if(! outcome) printf("equal\n");
    else if(outcome < 0) printf("first less than second\n");
    else printf("first greater than second\n");
}
```

出力

```
equal
```

関連項目

strcmp(), **strncmp**(), **strncasecmp**().

strcat

構文

#include <**string.h**>
char *strcat(char * restrict *s1***, const char * restrict s2);**

目的

文字列のコピーを別の文字列に追加します。

戻り値

strcat() は値 **s1** を返します。

パラメータ

s1 コピー先文字列へのポインタ。
s2 コピー元文字列へのポインタ。

説明

strcat() 関数は、**s2** が指す文字列のコピーを (終端の null 文字を含めて)、**s1** が指す文字列の末尾に追加します。**s2** の最初の文字は **s1** の末尾にある null 文字を上書きします。重複するオブジェクト間でコピーが実行された場合、動作は不定です。

例

出力

関連項目

strconcat(), **strjoin**(), **strncat**().

strchr

構文

#include <**string.h**>
char strchr(const char **s***,* **int** *c***)**

目的

文字列内で文字の最初の出現箇所を検索します。

戻り値

strchr() 関数は、検出された文字へのポインタを返します。または、文字列内に文字が出現しなかった場合は、null ポインタを返します。

パラメータ

s 検索対象となる文字列へのポインタ。
c 検索する文字。

説明

strchr() 関数は、**s** が指す文字列の **c** (**char** に変換される) の最初の出現箇所を検索します。終端の null 文字は文字列の一部と見なされます。

例

```
/* a sample program that uses strchr to return the remaining
string after and including the first space */
#include <stdio.h>
#include <string.h>

int main() {
    char *p;
    int c;

    c = ' ';
    p = "this This is a test";
    printf("Original string: %s\n", p);
    printf("Character to search for: '%c'\n", (char)c);
    p = strchr(p,c);
    printf("After: %s\n",p);
}
```

出力

```
Original string: this This is a test
Character to search for: ' '
After:  This is a test
```

関連項目

strcmp

構文

#include <**string.h**>
int strcmp(const char **s1***,* **const char** **s2***);*

目的

2 つの文字列を比較します。

戻り値

strcmp() 関数は、**s1** が指す文字列が、**s2** が指す文字列より大きいか、等しいか、小さいかに応じて、ゼロより大きい整数、ゼロに等しい整数、またはゼロより小さい整数を返します。ゼロ以外の戻り値の符号は、比較対象文字列で異なる最初のバイトどうしの値の差の符号によって決まります。

パラメータ

s1 比較対象の一方である文字列へのポインタ。
s2 比較対象の他方である文字列へのポインタ。

説明

strcmp() 関数は、**s1** が指す文字列と **s2** が指す文字列を比較します。

例

出力

関連項目

strcoll

構文

#include <**string.h**>
int strcoll(const char **s1***, **const char** **s2***);*

パラメータ

2 つの文字列を比較します。

戻り値

strcoll() 関数は、**s1** が指す文字列が、**s2** が指す文字列より大きいか、等しいか、小さいかに応じて、ゼロより大きい整数、ゼロに等しい整数、またはゼロより小さい整数を返します。両方の文字列は現在のロケールに応じて解釈されます。

パラメータ

s1 比較対象の一方である文字列へのポインタ。
s2 比較対象の他方である文字列へのポインタ。

説明

strcoll() 関数は、s1 が指す文字列と s2 が指す文字列を比較します。**s1** と **s2** は両方とも、現在のロケールの **LC_COLLATE** カテゴリに応じて解釈されます。

例

出力

関連項目

strconcat

構文

#include <**string.h**>
char *strconcat(const char * string1, ...);

目的

文字列を連結します。

戻り値

strconcat() は、動的に割り当てられたメモリを持つ文字列を返します。

パラメータ

string1 連結される文字列へのポインタ。

説明

strconcat() 関数は、引数リストのすべての文字列を連結し、動的にメモリが割り当てられる戻り値の文字列に結果を格納します。動的に割り当てられたメモリは利用者が解放する必要があります。

例

```
#include <string.h>
#include <stdarg.h>

int main() {
  char *buffer;
  char test1[90] = "abcd";

  buffer = strconcat(test1, "test2", "test3");
  printf("strconcat=%s\n",buffer);
  printf("test1=%s\n",test1);
  free (buffer);
  return 0;
}
```

出力

```
strconcat=abcdtest2test3
test1=abcd
```

関連項目

strcat(), **strncat()**, **strjoin()**.

strcpy

構文

#include <**string.h**>
char *strcpy(char * restrict *s1***, const char * restrict** *s2***);**

目的

文字列を配列にコピーします。

戻り値

strcpy() は値 **s1** を返します。

パラメータ

s1 コピー先配列へのポインタ。
s2 コピー元配列へのポインタ。

説明

strcpy() 関数は、**s2** が指す文字列を (終端の null 文字を含めて)、**s1** が指す配列にコピーします。重複するオブジェクト間でコピーが実行された場合、動作は不定です。

例

出力

関連項目

strcspn

構文

#include <**string.h**>
size_t *strcspn(const char **s1***, const char** **s2***);**

目的

文字列で、指定された文字列内の文字を含まない、先頭からのセグメントの最大の長さを計算します。

戻り値

strcspn() 関数は、セグメントの長さを返します。

パラメータ

s1 検索対象となる文字列へのポインタ。
s2 存在しないことを検索する文字列へのポインタ。

説明

strcspn() 関数は、**s1** が指す文字列で、**s2** が指す文字列に含まれない文字だけで構成される先頭からのセグメントの最大の長さを計算します。

例

```
/* a sample program that prints the number 8 which is
the index of the first charcacter in first string. */
#include <stdio.h>
#include <string.h>

int main() {
    int len;

    len = strcspn("this is a test","ab");
    printf("%d\n",len);
}
```

出力[1]

```
8
```

関連項目

[1]訳注：strcspn() 関数は大文字と小文字とを区別します。たとえば `strcspn("This is a test", "Tt")` は値ゼロを返します (最初の文字が`"T"`であるため)。`"T"`を削除して `strcspn("This is a test", "t")` とすると値 10 (文字列`"This is a "`のバイト数) が返されます。

strdup

構文

#include <**string.h**>
char *strdup(const char * *s***);**

目的

この関数は文字列を複製します。

戻り値

strdup() は、複製された文字列へのポインタを返します。

パラメータ

s 複製する文字列へのポインタ。

説明

strdup() 関数は、*s* が指す文字列の複製を作成し、**malloc** を使用して取得したメモリに格納します。

例

```
/* a sample program that displays a string. */
#include <stdio.h>
#include <string.h>

int main() {
    char str[80],*p;

    strcpy(str,"this is a test");
    p = strdup(str);   /* copy the content of str */
    if(p) {
      printf("%s\n",p);
      free(p);
    }
    else {
      printf("strdup() failed\n");
    }
}
```

出力

```
this is a test
```

関連項目

strerror

構文

#include <**string.h**>
char *strerror(int *errnum***);**

目的

errnum の値をマップします。

戻り値

strerror() 関数は、内容がロケール固有である文字列へのポインタを返します。ポイントされた配列をプログラムで変更することはできませんが、 **strerror ()** 関数への以降の呼び出しで上書きすることはできます。

パラメータ

errnum エラー番号。

説明

strerror() 関数は、**errnum** 内の数値をメッセージ文字列にマップします。通常、**errnum** の値は **errno** から取得されますが、 **strerror** は **int** 型の任意の値をメッセージにマップします。

実装では、ライブラリ関数による **strerror** 関数の呼び出しが存在しないかのように動作する必要があります。

例

出力

関連項目

strgetc

構文

#include <**string.h**>
int strgetc(**string_t** & *s*, **int** *i*);

目的

文字列の指定した位置の文字を取得します。

戻り値

この関数は文字を返します。

パラメータ

s 文字列への参照。
i 文字列内で取得される文字位置を示す整数。

説明

この関数は、文字列 *s* の *(i+1)* 番目の文字を返します。この関数は Ch の **string_t** に使用します。

例

```
/** access element of a string */
#include <stdio.h>
#include <string.h>

int main() {
  string_t s = "ABCD";

  printf("strgetc(s, 1) = %c \n", strgetc(s, 1));
  strputc(s, 2, 'm');
  printf("s = %s\n", s);
}
```

出力

```
strgetc(s, 1) = B
s = ABmD
```

関連項目

strputc().

strjoin

Synopsis
#include <string.h>
char *strjoin(const char * separator, ...);

目的
文字列を結合して、指定されたデリミタ文字列によって分離された文字列を作成します。

戻り値
strjoin() は、動的に割り当てられたメモリを持つ文字列を返します。

パラメータ
separator デリミタ文字列へのポインタ。

説明
strjoin() 関数は、引数リストのすべての文字列を結合し、動的にメモリが割り当てられる戻り値の文字列に結果を格納します。戻り値の文字列は、1 番目の引数 **separator** で指定されたデリミタによって分離されます。動的に割り当てられたメモリは利用者が解放する必要があります。

例

```
#include <string.h>
#include <stdarg.h>

int main() {
  char *buffer;

  buffer = strjoin("+", "test1", "test2", "test3");
  printf("strjoin =%s\n",buffer);
  free (buffer);
  return 0;
}
```

出力

```
strjoin =test1+test2+test3
```

関連項目
strcat(), **strncat()**, **strconcat()**.

strlen

構文
#include <**string.h**>
size_t strlen(const char *s**);**

目的
文字列の長さを計算します。

戻り値
strlen() 関数は、終端の null 文字の前にある文字数を返します。

パラメータ
s 文字列へのポインタ。

説明
strlen() 関数は、**s** が指す文字列の長さを計算します。

例
出力
関連項目

strncasecmp

構文

#include <**string.h**>
int strncasecmp(const char **s1***,* **const char** **s2***,* **size_t** *n***);**

目的

2 つの配列間で文字を比較します。

戻り値

strncasecmp() 関数は、**s1** が指す配列が、**s2** が指す配列より大きいか、等しいか、小さいかに応じて、ゼロより大きい整数、ゼロに等しい整数、またはゼロより小さい整数を返します。

パラメータ

s1 比較対象の一方である文字列へのポインタ。
s2 比較対象の他方である文字列へのポインタ。
n 比較する文字数。

説明

strncasecmp() 関数は、**s1** が指す配列から最大 **n** 個の文字 (null 文字に続く文字は比較されません) と、**s2** が指す配列を比較します。引数を比較する前に、 **tolower()** によってすべての文字が小文字に変換されます。

例

```
/* a sample program that shows the outcome of a comparison
of its two strings.*/
#include <stdio.h>
#include <string.h>

int main() {
    char s1[] = "abcdefG8";
    char s2[] = "abcdefg";
    int outcome;
    int l1,l2;
    int len;

    len = (l1 = strlen(s1)) < (l2 = strlen(s2)) ? l1:l2;
    outcome = strncasecmp(s1,s2,len);
    if(! outcome) printf("equal\n");
    else if(outcome < 0) printf("first less than second\n");
    else printf("first greater than second\n");
}
```

出力

```
equal
```

関連項目

strncat

構文

#include <**string.h**>
char *strncat(char * restrict *s1***, const char * restrict** *s2***, size_t** *n***);**

目的

文字列を文字列に追加します。

戻り値

strncat() は値 **s1** を返します。

パラメータ

s1 追加先文字列へのポインタ。
s2 追加元文字列へのポインタ。
n 追加する文字数。

説明

strncat() 関数は、**s2** が指す配列から最大 **n** 個の文字 (null 文字とそれに続く文字は追加されません) を、**s1** が指す文字列の終わりに追加します。**s2** の最初の文字は、**s1** の末尾にある null 文字を上書きします。終端の null 文字は常に結果に追加されます。重複するオブジェクト間でコピーが実行された場合、動作は不定です。

例

出力

関連項目

strcat(), **strconcat**(), **strjoin**().

strncmp

構文

#include <**string.h**>
int strncmp(const char ******s1***,* **const char** ******s2***,* **size_t** *n***);**

目的

2 つの配列間で文字を比較します。

戻り値

strncmp() 関数は、**s1** が指す配列が、**s2** が指す配列がより大きいか、等しいか、小さいかに応じて、ゼロより大きい整数、ゼロに等しい整数、またはゼロより小さい整数を返します。

パラメータ

s1 比較対象の一方である文字列へのポインタ。
s2 比較対象の他方である文字列へのポインタ。
n 比較する文字数。

説明

strncmp() 関数は、**s1** が指す配列から最大 **n** 個以下の文字 (null 文字に続く文字は比較されません) と、**s2** が指す配列を比較します。

例

```
/* a sample program that shows the outcome of a comparison
of its two strings.*/
#include <stdio.h>
#include <string.h>

int main() {
    char s1[] = "abcdefghijkl";
    char s2[] = "abcdefkl";
    int outcome;
    int l1,l2;
    int len;

    len = (l1 = strlen(s1)) < (l2 = strlen(s2)) ? l1:l2;
    outcome = strncmp(s1,s2,len);
    if(! outcome) printf("equal\n");
    else if(outcome < 0) printf("first less than second\n");
    else printf("first greater than second\n");
}
```

出力

```
first less than second
```

関連項目

strncpy

構文

#include <**string.h**>
char *strncpy(char * restrict *s1***, const char * restrict** *s2***, size_t** *n***);**

目的

配列の文字を別の配列にコピーします。

戻り値

strncpy() は値 **s1** を返します。

パラメータ

s1 コピー先文字列へのポインタ。
s2 コピー元文字列へのポインタ。
n コピーする文字数。

説明

strncpy() 関数は、**s2** が指す配列から最大 **n** 個の文字 (null 文字に続く文字はコピーされません) を、**s1** が指す配列にコピーします。重複するオブジェクト間でコピーが実行された場合、動作は不定です。

s2 が指す配列が **n** 個の文字より短い文字列である場合、**n** 個分の文字がすべて書き込まれるまで、**s1** が指す配列のコピーに null 文字が追加されます。

例

出力

関連項目

strpbrk

構文

#include <**string.h**>
char *strpbrk(const char **s1***, **const char** **s2***);**

目的

文字列内の文字の最初の出現箇所を別の文字列内で検索します。

戻り値

strpbrk() 関数は、文字へのポインタを返します。または、**s1** 内に **s2** の文字が出現しなかった場合は、null ポインタを返します。

パラメータ

s1 検索する文字列へのポインタ。
s2 検索に使用される文字セットへのポインタ。

説明

strpbrk() 関数は、**s1** が指す文字列で、**s2** が指す文字列に含まれるいずれかの文字の最初の出現箇所を検索します。

例

```
/* a sample program that prints the message 's is a test'
on the screen. */
#include <stdio.h>
#include <string.h>

int main() {
    char *p;

    p = strpbrk("this is a test"," absj");
    printf("%s\n",p);
}
```

出力

```
s is a test
```

関連項目

strputc

構文
#include <**string.h**>
int strputc(**string_t** &*s*, **int** *i*, **char** *c*);

目的
文字列の指定した位置の文字を別の文字に置き換えます。

戻り値
この関数 () は、成功すると 0 を返し、失敗すると-1 を返します。

パラメータ
s 文字列への参照。
i 置換される文字の位置を示す入力整数。
c 文字列内の文字を置き換えるために使用される入力文字。

説明
この関数は、文字 *c* を含む文字列 *s* の *(i+1)* 番目の文字を置き換えます。この関数は Ch の **string_t** に使用します。

例
strgetc を参照してください。

関連項目
strgetc().

strrchr

構文

#include <**string.h**>
char *strrchr(const char ******s***, int** *c***);**

目的

文字列内での文字の最後の出現箇所を検索します。

戻り値

strrchr() 関数は、文字へのポインタを返します。または、文字列内に **c** が出現しなかった場合は、null ポインタを返します。

パラメータ

s 検索対象となる文字列へのポインタ。
c 検索する文字。

説明

strrchr() 関数は、**s** が指す文字列で **c(char** に変換される**)** の最後の出現箇所を検索します。終端の null 文字は、文字列の一部と見なされます。

例

```
/* a sample program that prints the message 'is a test'
on the screen. */
#include <stdio.h>
#include <string.h>

int main() {
    char *p;

    p = strrchr("this is a test",'i');
    printf("%s\n",p);
}
```

出力

```
is a test
```

関連項目

strrep

構文

#include <**string.h**>
string_t strrep(string_t *s1*, **string_t** *s2*, **string_t** *s3*);

目的

別の文字列内で文字列を検索し、置き換えます。

戻り値

この関数は、一致した文字列がすべて置き換えられた新しい文字列を返します。

パラメータ

s1 文字列 *s2* が検索、置換される文字列。
s2 検索対象の文字列。
s3 文字列 *s2* の置換に使用される文字列。

説明

この関数は、文字列 *s1* 内で文字列 *s2* を検索し、文字列 *s3* に置換します。置換された文字列が返され、文字列 *s1* は変更されません。

例

```
#include <stdio.h>
#include <string.h>

int main() {
   string_t s, s1 = "He is smart, but you are smarter",
             s2 ="smart", s3="great", s4="tall";

   printf("s1 = %s\n", s1);
   s = strrep(s1, s2, s3);
   printf("s1 = %s\n", s1);
   printf("s = %s\n", s);
   s = strrep(s1, s2, s3);
   printf("s1 = %s\n", s1);
   printf("s = %s\n", s);
   s = strrep(s1, s2, s4);
   printf("s1 = %s\n", s1);
   printf("s = %s\n", s);
}
```

出力

```
s1 = He is smart, but you are smarter
s1 = He is smart, but you are smarter
s = He is great, but you are greater
s1 = He is smart, but you are smarter
```

```
s = He is great, but you are greater
s1 = He is smart, but you are smarter
s = He is tall, but you are taller
```

関連項目

strgetc().

strspn

構文

#include <**string.h**>
size_t strspn(const char **s1***,* **const char** **s2***);*

目的

文字列で、指定された文字列内の文字だけから構成される、先頭からのセグメントの最大の長さを計算します。

戻り値

strspn() 関数は、セグメントの長さを返します。

パラメータ

s1 検索対象となる文字列へのポインタ。
s2 構成対象となる文字セットへのポインタ。

説明

strspn() 関数は、**s1** が指す文字列で、**s2** が指す文字列に含まれる文字だけで構成される先頭からのセグメントの最大の長さを計算します。

例

```
/* a sample program that prints the index of the first
character in the string pointed to by str1 that does not
match any of the characters in str2.*/
#include <stdio.h>
#include <string.h>

int main() {
    unsigned int len;

    len = strspn("this is a test","siht ");
    printf("%d\n",len);
}
```

出力[2]

```
8
```

関連項目

[2]訳注：strspn() 関数は大文字と小文字とを区別します。たとえば `strspn("This is a test", "siht")` は値ゼロを返します (最初の文字が`"T"`であるため)。この場合にも同じ値 8 を得るには、`"T"`を追加してたとえば `strspn("This is a test", "siTht")` としなければなりません (`"T"`の場所は問いません)。

strstr

構文

#include <**string.h**>
char *strstr(const char **s1***, **const char** **s2***);**

目的

文字列の最初の出現箇所を別の文字列内で検索します。

戻り値

strstr() 関数は、検出された文字列へのポインタを返します。または、文字列が見つからない場合は、null ポインタを返します。**s2** が長さゼロの文字列を指している場合、この関数は **s1** を返します。

パラメータ

s1 検索される文字列へのポインタ。
s2 検索に使用される文字列へのポインタ。

説明

strstr() 関数は、**s1** が指す文字列で、**s2** が指す文字列に含まれる文字のシーケンス (終端の null 文字を含まない) の最初の出現箇所を検索します。

例

```
/* a sample program that prints 'is is a test'. */
#include <stdio.h>
#include <string.h>

int main() {
    char *p;

    p = strstr("this is a test","is");
    printf("%s\n",p);
}
```

出力

```
is is a test
```

関連項目

strtok_r

構文

#include <**string.h**>
char *strtok_r(char * *s***, const char *** *sep***, char **** *lasts***);**

目的

指定されたセパレータで区切られたすべての文字列を検索します。

戻り値

strtok_r() 関数は、検出されたトークンへのポインタを返します。または、トークンが見つからない場合は、null ポインタを返します。

パラメータ

s 検索される文字列へのポインタ。
sep セパレータとして使用される文字列へのポインタ。
lasts ユーザ指定のポインタへのポインタ。

説明

この関数は、null で終わる文字列 *s* を、文字列 *sep* に含まれる 1 つ以上の文字で区切ったゼロ個以上のテキストトークンのシーケンスと見なします。*lasts* は、同じ文字列のスキャンを続行するために/strtokr/に必要な保存済み情報を指します。

最初の呼び出しでは、*s* は null で終わる文字列を指し、*sep* は null で終わる区切り文字の文字列を指します。*lasts* は無視されます。この関数は、トークンの最初の文字へのポインタを返し、*s* に null 文字とそれに続く戻り値のトークンを書き込み、lasts が指すポインタを更新します。

2 回目以降の呼び出しでは、*s* は **NULL** ポインタで、*lasts* は前の呼び出しと変わりません。これにより、以降の呼び出しで文字列 s を処理し、処理対象のトークンがなくなるまで連続するトークンを返します。区切り文字の文字列 *sep* は呼び出しごとに変更しても構いません。*s* にトークンが残っていない場合は、**NULL** ポインタが返されます。

例

```
/* a sample program that tokenizes the string, "The summer soldirer,
the sunshine patriot" with spaces and commas being the delimiters.
The output is The|summer|soldier|the sunshine|patriot. */
#include <stdio.h>
#include <string.h>

int main() {
    char *token, *str1, *delimit;
    char *endptr = NULL;
```

```
    str1 = "The summer soldier, the sunshine patriot";
    delimit = " ,";
    for(token = strtok_r(str1, delimit, &endptr); token!= NULL;
        token = strtok_r(NULL, delimit, &endptr)) {
        printf("token = %s\n", token);
    }
}
```

出力

```
token = The
token = summer
token = soldier
token = the
token = sunshine
token = patriot
```

関連項目

strtok

構文

#include <**string.h**>
char *strtok(char * restrict *s1***, const char * restrict** *s2***);**

目的

文字列をトークンのシーケンスに分割します。

戻り値

strtok() 関数は、トークンの最初の文字へのポインタを返します。または、トークンが見つからない場合は、null ポインタを返します。

パラメータ

s1 検索される文字列へのポインタ。
s2 セパレータとして使用される文字列へのポインタ。

説明

strtok() 関数の連続した呼び出しによって、**s1** が指す文字列をトークンのシーケンスに分割します。各トークンは、**s2** が指す文字列の文字で区切られます。連続呼び出しの初回では最初の引数に null 以外を指定し、連続する後続の呼び出しでは最初の引数に null を指定します。**s2** が指す区切り文字の文字列は、呼び出しごとに変更しても構いません。

連続呼び出しの初回では、**s1** が指す文字列で、**s2** が指す現在の区切り文字の文字列に含まれない最初の文字を検索します。このような文字が見つからない場合、**s1** が指す文字列にトークンはないため、**strtok** 関数は null ポインタを返します。このような文字が見つかった場合、その文字が最初のトークンの先頭です。
strtok() 関数は、そこから現在の区切り文字の文字列に含まれている文字を検索します。このような文字が見つからない場合、**s1** が指す文字列の末尾までが現在のトークンであり、後続のトークンの検索では null ポインタが返されます。このような文字が見つかった場合、その文字は null 文字で上書きされ、現在のトークンが終了します。 **strtok** 関数は、トークンの次の検索を開始する次の文字へのポインタを保存します。

後続の各呼び出しでは最初の引数の値として null ポインタを指定し、保存したポインタから検索を開始し、上述したように動作します。

実装では、ライブラリ関数による **strtok** 関数の呼び出しが存在しないかのように動作する必要があります。

例

```
/* a sample program that tokenizes the string, "The summer soldirer,
the sunshine patriot" with spaces and commas being the delimiters.*/
#include <stdio.h>
#include <string.h>

int main() {
    char *token, *str1, *delimit;

    str1 = "The summer soldier, the sunshine patriot";
    delimit = " ,";
    for(token = strtok(str1, delimit); token!= NULL;
        token = strtok(NULL, delimit)) {
        printf("token = %s\n", token);
    }
}
```

出力

```
token = The
token = summer
token = soldier
token = the
token = sunshine
token = patriot
```

関連項目

strxfrm

構文

#include <**string.h**>
size_t strxfrm(char * restrict *s1***, const char * restrict** *s2***, size_t** *n***);**

目的

ロケール固有情報に基づいて文字列を変換し、配列内に配置します。

戻り値

strxfrm() 関数は、変換された文字列の長さ (終端の null 文字を含まない) を返します。戻り値が **n** 以上である場合、**s1** が指す配列の内容は不定です。

パラメータ

s1 変換先文字列へのポインタ。
s2 変換する文字列へのポインタ。
n 変換する文字数。

説明

strxfrm() 関数は、**s2** が指す文字列を変換し、**s1** が指す配列に結果の文字列を配置します。変換は現在のロケールの **LC_COLLATE** カテゴリの設定にしたがって、次のように行われます。つまり、2 つの文字列それぞれに対して **strxfrm()** 関数を適用した後、**strcmp** 関数を変換後の 2 つの文字列に適用すると、変換前の同じ 2 つの文字列に **strcoll** 関数を適用した結果と同一の結果が得られます。**s1** が指す結果の配列には終端の null 文字も含め、最大 **n** 個の文字が配置されます。重複するオブジェクト間でコピーが実行された場合、動作は不定です。
“C”ロケールでは文字セットの文字順と辞書順とは同じですので、文字を辞書順で比較する場合にも **strxfrm()** 関数を呼び出して変換する必要はありません。この関数は、ヨーロッパのあるロケールのように文字順と辞書順とが異なる場合に、辞書式の比較をする場合などに使用します。そのような場合に直接 **strcoll()** 関数を呼び出しても構いません。

例

出力

関連項目

第20章　日付と時間の関数 — <time.h>

ヘッダー **time.h** では 1 つのマクロを定義し、時間を操作する複数の型と関数を宣言します。多くの関数では、現在の日付 (グレゴリオ暦に従う) と時刻を示すカレンダー時刻を扱います。特定のタイムゾーンを表すカレンダー時刻であるローカル時刻や、ローカル時刻を決定するアルゴリズムで一時的に変更される夏時間を扱う関数もあります。ローカルタイムゾーンと夏時間は、実装で定義されます。

定義されているマクロは **CLOCKS_PER_SEC** です。これは以下に示す **clock_t** 型の定数式に展開され、1 秒間あたりのタイマ刻みの数を表します。

宣言されている型は **size_t**、
　clock_t
および
　time_t
です。これらは時刻を表現できる算術型です。また、
　struct tm
は、詳細時刻と呼ばれるカレンダー時刻の要素を保持します。

tm 構造体には、次のメンバが含まれます。メンバの意味と正常な範囲をコメント内に示しています。

```
int tm_sec; // 毎分の後の秒—[0,60]
int tm_min; // 毎時の後の分—[0,59]
int tm_hour; // 真夜中以来の時間—[0,23]
int tm_mday; // 日付—[1,31]
int tm_mon; // 1 月以来の月—[0,11]
int tm_year; // 1900 年以来の年
 int tm_wday; // 日曜日以来の日数—[0,6]
int tm_yday; // 1 月 1 日以来の日数—[0,365]
int tm_isdst; // 夏時間フラグ
```

tm_isdst の値は夏時間が有効な場合は正の数、夏時間が無効な場合はゼロ、情報が取得できない場合は負の数です。

パブリックデータ

なし。

関数

time ヘッダーを使用して次の関数が実装されています。

関数	説明
asctime()	詳細時刻を文字列に変換します。
clock()	消費されたプロセッサ時間を取得します。
ctime()	カレンダー時刻をローカル時刻に変換し、その後文字列に変換します。
difftime()	2 つのカレンダー時刻の差を計算します。
gmtime()	カレンダー時刻を UTC で表した詳細時刻に変換します。
localtime()	カレンダー時刻をローカル時刻で表した詳細時刻に変換します。
mktime()	詳細時刻をカレンダー時刻に変換します。
strftime()	指定された日時を書式化して返します。
time()	現在のカレンダー時刻を取得します。

マクロ

time ヘッダーでは、次のマクロが定義されています。

マクロ	説明
CLOCKS_PER_SEC	1 秒間あたりのタイマ刻みの数を保持します。

宣言型

time ヘッダーでは、次の型が宣言されています。

宣言型	説明
size_t	符号なし整数。
clock_t	時刻を表現できる算術型。
time_t	時刻を表現できる算術型。
struct tm	各種の日時要素を保持する構造体。

移植性
このヘッダーには移植性に関する既知の問題はありません。

asctime

構文

#include <**time.h**>
char * asctime(const struct tm ******timeptr***);**

目的

詳細時刻を文字列型に変換します。

戻り値

asctime() 関数は文字列へのポインタを返します。

パラメータ

timeptr tm 型構造体へのポインタ

説明

asctime() 関数は、*timeptr* が指す構造体の詳細時刻を、

```
char *asctime(const struct tm *timeptr)
{
     static const char wday_name[7][3] = {
          "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
     };
     static const char mon_name[12][3] = {
          "Jan", "Feb", "Mar", "Apr", "May", "Jun",
          "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
     };
     static char result[26];

     sprintf(result, "%.3s %.3s%3d %.2d:%.2d%.2d %d\n",
          wday_name[timeptr->tm_wday],
          mon_name[timeptr->tm_mon],
          timeptr->tm_mday, timeptr->tm_hour,
          timeptr->tm_min, timeptr->tm_sec,
          1900 + timeptr->tm_year;
     return result;
}
```

と同等のアルゴリズムを使用して、

```
Mon Mar 10 01:03:52 2008\n\0
```

の形式の文字列に変換します。

例 1

```
/* a sample program that return local time. */
```

```
#include <stdio.h>
#include <time.h>

int main() {
    struct tm *ptr;
    time_t lt;

    lt = time(NULL);       //return system time
    printf("time(NULL) = %d\n", lt);
    ptr = localtime(&lt);  // return time in the form of tm structure
    printf(asctime(ptr));
}
```

出力

```
time(NULL) = 941830207
Fri Nov  5 14:30:07 1999
```

例 2

```
/* a sample program that return local time. */
#include <stdio.h>
#include <time.h>
#define SIZE 26

int main() {
    struct tm *ptr;
    time_t lt;
    char buf[26];

    lt = time(NULL);       //return system time
    printf("time(NULL) = %d\n", lt);
    ptr = localtime(&lt);  // return time in the form of tm structure
    asctime_r(ptr,buf,SIZE);
    printf(buf);
}
```

出力

```
time(NULL) = 941830207
Fri Nov  5 14:30:07 1999
```

関連項目

clock

構文

#include <**time.h**>
clock_t clock(*void***);**

目的

消費されたプロセッサ時間を取得します。

戻り値

clock() 関数は、プログラムが起動されてからこの関数が呼び出されるまでに消費したプロセッサ時間に最も近い実装の近似値を返します。秒単位で時刻を判断するには、 **clock()** 関数からの戻り値をマクロ **CLOCKS_PER_SEC** の値で除算します。消費したプロセッサ時間が取得できない場合、またはその値が表現できない場合、関数は値 **(clock_t) – 1** を返します。

パラメータ

引数なし。

説明

clock() 関数は、消費したプロセッサ時間を取得します。

例

```
/* a sample program that displays the current execution time. */
#include <stdio.h>
#include <time.h>
void elapsed_time(void);

int main() {
    int i;

    clock();    /// first call clock
    for(i = 0; i<100000;i++) {};
    elapsed_time();
}

void elapsed_time(void) {
    clock_t time;

    time = clock();
    printf("Elapsed time: %d micro seconds or %f seconds.\n",time,
          (double)time/CLOCKS_PER_SEC);  //  return the time between first call
                                         // and this call .
}
```

出力

```
Elapsed time: 30000 microsecs.
```

ctime

構文

#include <**time.h**>
char * ctime(const time_t ******timer***);**

目的

カレンダー時刻をローカル時刻に変換し、その後文字列に変換します。

戻り値

ctime() 関数は、引数に詳細時刻を持つ **asctime()** 関数から返されたポインタを返します。

パラメータ

timer カレンダー時刻の保持に使用される変数のポインタ。

説明

ctime() 関数は、 **timer** が指すカレンダー時刻を文字列の形式のローカル時刻に変換します。これは次のコードと同等です。

asctime(localtime(timer))

例

```
/* a sample program that return local time. */
#include <stdio.h>
#include <time.h>

int main() {
    time_t *timer;

    timer = malloc(sizeof(time_t));
    *timer = time(NULL);       //return system time
    printf("%s", ctime(timer));
}
```

出力

```
Fri Nov  5 14:30:09 1999
```

関連項目

difftime

構文

#include <**time.h**>
double difftime(time_t *time1*, **time_t** *time0***);**

目的

2 つのカレンダー時刻の差を計算します。

戻り値

difftime() 関数は、秒単位で表される差を double 型で返します。

パラメータ

time1 終了カレンダー時刻。
time0 開始カレンダー時刻。

説明

difftime() 関数は、カレンダー時刻 *time0* からカレンダー時刻 *time1* までの経過時間を計算します。

例

```
/* a sample program that returns the difference in
seconds between two calendar times. */
#include <stdio.h>
#include <time.h>

int main() {
    time_t time0, time1;
    int i;

    time0 = time(NULL);       /* return system time */
    for(i = 0;i<100000;i++) {};
    time1 = time(NULL);
    printf("the 100000 loops time is %f sec\n",difftime(time1,time0));
}
```

出力

```
the 100000 loops time is 8.000000 sec
```

関連項目

gmtime

構文

#include <**time.h**>
struct tm * gmtime(const time_t ******timer***);**

目的

カレンダー時刻を UTC で表した詳細時刻に変換します。

戻り値

gmtime() 関数は、詳細時刻へのポインタを返すか、または、指定された時刻を UTC (世界協定時) に変換できない場合は null ポインタを返します。

パラメータ

timer カレンダー時刻の保持に使用される変数のポインタ。

説明

gmtime() 関数は、*timer* が指すカレンダー時刻を UTC で表した詳細時刻に変換します。

例

```
/* a sample program that prints both the local time and
the UTC of the system. */
#include <stdio.h>
#include <time.h>

int main() {
    struct tm *local_time, *gm;
    time_t t;

    t = time(NULL);
    local_time = localtime(&t);
    printf("Local time and date: %s\n",asctime(local_time));
    gm = gmtime(&t);
    printf("Coordinated Universal Time and date: %s\n",asctime(gm));
}
```

出力

```
Local time and date: Fri Nov  5 14:30:18 1999

Coordinated Universal Time and date: Fri Nov  5 19:30:18 1999
```

関連項目

localtime

構文

#include <**time.h**>
struct tm * localtime(const time_t ******timer***);**

目的

カレンダー時刻をローカル時刻で表した詳細時刻に変換します。

戻り値

localtime() 関数は、詳細時刻へのポインタを返すか、または、指定された時刻をローカル時刻に変換できない場合は null ポインタを返します。

パラメータ

timer カレンダー時刻の保持に使用される変数のポインタ。

説明

localtime() 関数は、*timer* が指すカレンダー時刻をローカル時刻で表した詳細時刻に変換します。

例

```
/* a sample program that prints both the local time and
the UTC of the system. */
#include <stdio.h>
#include <time.h>

int main() {
    struct tm *local_time, *gm;
    time_t t;

    t = time(NULL);
    local_time = localtime(&t);
    printf("Local time and date: %s\n",asctime(local_time));
    gm = gmtime(&t);
    printf("Coordinated Universal Time and date: %s\n",asctime(gm));
}
```

出力

```
Local time and date: Fri Nov  5 14:30:19 1999

Coordinated Universal Time and date: Fri Nov  5 19:30:19 1999
```

関連項目

mktime

構文

#include <**time.h**>
time_t mktime(struct tm*timeptr***);**

目的

詳細時刻をカレンダー時刻に変換します。

戻り値

mktime() 関数は、 **time_t** 型の値にエンコードされた特定のカレンダー時刻を返します。カレンダー時刻を表現できない場合、関数は値 **(time_t) – 1** を返します。

パラメータ

timeptr tm 型の構造体へのポインタ

説明

mktime() 関数は、*timeptr* が指す構造体の詳細時刻 (ローカル時刻で表される) を **time()** 関数が返す値と同じエンコードのカレンダー時刻の値に変換します。構造体の元の **tm_wday** および **tm_yday** 要素は無視され、他の要素の元の値は上述の範囲には制限されません。正常に終了した場合は、構造体の **tm_wday** および **tm_yday** 要素の値が適切に設定され、他の要素は指定されたカレンダー時間を表すように設定されます。ただし、このときの他の要素の値は上述の範囲内に強制されます。**tm_mday** の最後の値は、 **tm_mon** および **tm_year** の決定後に設定されます。

呼び出しに成功すると、 **struct tm** 値の結果を使用した **mktime()** 関数の 2 回目の呼び出しは、**struct tm** 値を変更せずに、常に 1 回目の呼び出しと同じ値を返します。さらに、正規化された時刻を **time_t** 値として正確に表現できる場合、正規化された詳細時刻と **localtime()** を呼び出して **mktime()** 関数の結果を変換して得られた詳細時刻は一致します。

例

```
/* a sample program that shows the use of mktime() */
#include <stdio.h>
#include <time.h>

int main() {
    struct tm t;
    time_t t_of_day;

    t.tm_year = 1999-1900;
    t.tm_mon = 6;
    t.tm_mday = 12;
    t.tm_hour = 2;
    t.tm_min = 30;
```

```
    t.tm_sec = 1;
    t.tm_isdst = 0;
    t_of_day = mktime(&t);
    printf(ctime(&t_of_day));

}
```

出力

```
Mon Jul 12 03:30:01 1999
```

strftime

構文

#include <**time.h**>
char strftime(char * restrict *s***, size_t** *maxsize***, const char * restrict** *format***, const struct tm * restrict** *timeptr***);**

目的

指定された日時を書式化して返します。

戻り値

終端の null 文字を含む結果の総文字数が *maxsize* 以下の場合、**strftime**() 関数は、*s* が指す配列に代入された終端の null 文字を含まない文字数を返します。それ以外の場合はゼロを返し、配列の内容は不定となります。

パラメータ

s 書式化された文字列を格納する配列へのポインタ。
maxsize 文字列の最大長。
format 書式文字列へのポインタ。
timeptr 変換する日時を保持する *tm* 構造体へのポインタ。

説明

strftime() 関数は、*s* が指す配列に文字を代入します。このとき、*format* が指す文字列の制御を受けます。入力形式は、初期シフト状態の開始と終了を持つ、マルチバイト文字のシーケンスです。*format* 文字列は、ゼロまたは 1 つ以上の変換指定子と通常のマルチバイト文字で構成されます。変換指定子は、**%**文字と、状況によってその後に続く修飾子文字 (以下に示す) の **E** または **O** で構成されます。変換指定子の後には、その動作を決定する文字が続きます。すべての通常マルチバイト文字 (終端の null 文字を含む) は変更されずに、配列にコピーされます。重複するオブジェクト間でコピーが行われる場合、動作は未定義です。*maxsize* 以下の文字が配列に代入されます。

各変換指定子は、以下のリストに示す適切な文字に置き換えられます。現在のロケールの **LC_TIME** カテゴリを使用して、**timeptr** が指すゼロまたは 1 つ以上の詳細時間の構造体メンバ値によって、適切な文字を決定します。指定したいずれかの値が正常範囲外である場合、保存される文字は未定義です。

%a は、ロケールの曜日の省略名称に置き換えられます。 [**tm_wday**]

%A は、ロケールの曜日の省略名称に置き換えられます。 [**tm_wday**]

%b は、ロケールの月の省略名称に置き換えられます。 [**tm_mon**]

%B は、ロケールの月の正式名称に置き換えられます。 [**tm_mon**]

%c は、ロケールの適切な日時表現に置き換えられます。

%d は、月の日付を示す 10 進数 (01〜31) に置き換えられます。**(01–31)**. [**tm_mday**]

%H は、時間 (24 時間表記) を示す 10 進数 (**(01–23)**) に置き換えられます。 [**tm_hour**]

%I は、時間 (12 時間表記) を示す 10 進数 (**(01–12)**) に置き換えられます。 [**tm_hour**]

%j は、年間の日にちを示す 10 進数 (**(001–366)**) に置き換えられます。 [**tm_yday**]

%m は、月を示す 10 進数 (**(01–12)**) に置き換えられます。 [**tm_mon**]

%M は、分を示す 10 進数 (**(00–59)**) に置き換えられます。 [**tm_min**]

%n は、改行文字に置き換えられます。

%p は、12 時間表記に関連付けたロケールの午前および午後指定に相当する値に置き換えられます。 [**tm_hour**]

%S は、秒を示す 10 進数 (**(00–60)**) に置き換えられます。 [**tm_sec**]

%t は、水平タブ文字に置き換えられます。

%U は、年間の週番号を示す 10 進数 (**(00–53)**) に置き換えられます。最初の日曜日を第 1 週の開始日とします。 [**tm_year**, **tm_wday**, **tm_yday**]

%w は、曜日を示す 10 進数 (**(0–6)**) に置き換えられます。日曜日が 0 です。[**tm_wday**]

%W は、年間の週番号を示す 10 進数 **((00–53))** に置き換えられます。最初の月曜日を第 1 週の開始日とします。 [**tm_year**, **tm_wday**, **tm_yday**]

%x は、ロケールの適切な日付表現に置き換えられます。

%X は、ロケールの適切な時間表現に置き換えられます。

%y は、年の下 2 桁を示す 10 進数 **((00–99))** に置き換えられます。 [**tm_year**]

%Y は、年を示す 10 進数 (**1997**) に置き換えられます。 [**tm_year**]

%z は、ISO 8601 形式の UTC との差分に置き換えられます。値の形式は "**-0430**" は、ISO 8601 形式の UTC との差分に置き換えられます。値の形式は文字は設定されません。 [**tm_isdst**]

%Z は、ロケールのタイムゾーンの名前または省略名称に置き換えられます。タイムゾーンを判定できない場合は、文字は設定されません。 [**tm_isdst**]

%% は、**%**に置き換えられます。

変換指定子が上記以外の場合、動作は未定義です。

"C" ロケールでは、修飾子 **E** および **O** は無視されて、次の指定子の文字列に交換されます。

%a **%A** の最初の 3 文字。

%A **"日曜日", "月曜日", ... ,"土曜日"** のいずれか。

%b **%B** の最初の 3 文字のいずれか。

%B **"1 月", "2 月", ... , "3 月"** のいずれか。

%c **"%A %B %d %T %Y"** に相当する値。

%p **“am”** または **“pm”** のいずれか。

%r **“%I:%M:%S:%p”** に相当する値。

%x **“%A:%B:%d:%Y”** に相当する値。

%X **“%T”** に相当する値。

%Z 実装定義による値。

例

```
/* a sample program that displays current time. */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    struct tm *ptr;
    time_t ltime;
    char str[80];

    ltime = time(NULL); /* get current calendar time */
    ptr = localtime(&ltime);  // return time in the form of tm structure
    strftime(str,80,"It is now %H:%M %p.",ptr);
    printf("%s\n",str);
}
```

出力

```
It is now 14:30 PM.
```

time

構文

#include <**time.h**>
time_t time(time_t *timer***);**

目的

現在のカレンダー時刻を取得します。

戻り値

time() 関数は、現在のカレンダー時刻に最も近い実装の近似値を返します。カレンダー時刻を取得できない場合は、値（**time_t**）`-1` が返されます。*timer* が null ポインタでない場合、戻り値は timer が指すオブジェクトにも代入されます。

パラメータ

timer カレンダー時刻の保持に使用される変数。

説明

time() 関数は、現在のカレンダー時刻を取得します。値のエンコードは未指定です。

例

```
See asctime().
```

出力

第21章　拡張マルチバイトおよびワイド文字関数 — ＜wchar.h＞

ヘッダー **wchar.h** は、4つのデータ型、1つのタグ、4つのマクロ、および多くの関数を宣言しています。

宣言されている型は、 **wchar_t** と 16章で説明されている **size_t** であり、

mbstate_t

は、マルチバイト文字とワイド文字のシーケンス間の変換に必要な変換状態情報を保存可能な配列以外のオブジェクト型であり、
22章で説明されている **wint_t**

および

struct tm

は、不完全な構造体型として宣言され、その内容は 20章で説明されています。
宣言されているマクロは以下のとおりです。

WCHAR_MAX

は、**wchar_t** 型のオブジェクトで表される最大値であり、

WCHAR_MIN

は、**wchar_t** 型のオブジェクトで表される最小値であり、そして

WEOF は 22章で説明されています。

宣言されている関数は、以下のようにグループ分けされています。

— ワイド文字またはマルチバイト文字、あるいはその両方の入出力を行う関数、

— ワイド文字の数値変換を提供する関数、

— 一般的なワイド文字操作を行う関数、

— ワイド文字の日付と時間の変換用関数、および

— マルチバイトおよびワイド文字シーケンス間の変換用の拡張機能を提供する関数。

明示的に説明されていないその他のケースでは、この節で説明されている関数の実行によりオーバーラップするオブジェクト間のコピーが行われた場合、その結果は不定になります。

Public データ
なし。

関数
wchar ヘッダーを用いて実装される関数は、以下のとおりです。

関数	説明
_wfopen()	ファイルを開きます。
btowc()	文字がマルチバイト文字であるかどうかを定めます。
fgetwc()	ワイド文字を取得し、位置を 1 つ進めます。
fgetws()	ストリームからワイド文字を読みます。
fputwc()	ワイド文字を出力ストリームに書き込みます。
fputws()	ワイド文字列をストリームに書き込みます。
fwide()	ストリームの方向を定めます。
getwc()	ワイド文字を取得して、ポインタを進めます。
getwchar()	ワイド文字を取得して、ファイル位置インジケータを進めます。
mbrlen()	マルチバイト文字におけるバイト数を定めます。
mbrtowc()	次のマルチバイト文字の完了に必要なバイト数を定めます。
mbsinit()	ポイントされるオブジェクトが初期変換状態を記述しているかどうかを定めます。
mbsrtowcs()	配列のマルチバイト文字シーケンスをワイド文字シーケンスに変換します。
putwc()	ワイド文字を出力ストリームに書き込みます。
putwchar()	ワイド文字を出力ストリームに書き込みます。
ungetwc()	ワイド文字を入力ストリームに戻します。
wcrtomb()	次のマルチバイト文字の表示に必要なバイト数を定めます。
wcscat()	1 つのワイド文字列のコピーを他のワイド文字列に追加します。
wcschr()	文字列内であるワイド文字の最初の出現箇所を返します。
wcscmp()	2 つのワイド文字列を比較します。
wcscoll()	2 つのワイド文字列を比較します。
wcscpy()	ワイド文字列を配列にコピーします。
wcscspn()	ワイド文字列で、先頭からのセグメントの最大の長さを計算します。
wcsftime()	ワイド文字を配列内に置きます。
wcslen()	ワイド文字列の長さを計算します。

wcsncat()	配列のワイド文字をワイド文字列に追加します。
wcsncmp()	2 つの配列のワイド文字を比較します。
wcsncpy()	ある配列のワイド文字を他の配列にコピーします。
wcspbrk()	文字列内のワイド文字の最初の出現箇所を別の文字列内で検索します。
wcsrchr()	ワイド文字列内でのワイド文字の最後の出現箇所を検索します。
wcsrtombs()	配列内のワイド文字シーケンスをマルチバイト文字シーケンスに変換します。
wcsspn()	ワイド文字列で、先頭からのセグメントの最大の長さを計算します。
wcsstr()	ワイド文字列内で検索された最初のシーケンスを他のワイド文字列内で検索します。
wcstod()	ワイド文字列の最初の部分を double、float、および long double に変換します。
wcstok()	ワイド文字列をトークンのシーケンスに分割します。
wcstol()	ワイド文字列を long int、long long int、unsigned long int、または unsigned long long int に変換します。
wcsxfrm()	ワイド文字列を変換し、配列内に配置します。
wctob()	文字が拡張文字セットのメンバかどうか定めます。
wmemchr()	ストリーム内で、指定されたワイド文字の最初の出現箇所を検索します。
wmemcmp()	2 つのオブジェクト内の最初のワイド文字を比較します。
wmemcpy()	オブジェクトのワイド文字を別のオブジェクトにコピーします。
wmemmove()	オブジェクトのワイド文字を別のオブジェクトにコピーします。
wmemset()	オブジェクトの指定された数の位置に単一の値をコピーします。

マクロ

wchar ヘッダーでは、以下のマクロが定義されています。

マクロ	説明
WCHAR_MAX	オブジェクト型 **wchar_t** で表現可能な最大値。
WCHAR_MIN	オブジェクト型 **wchar_t** で表現可能な最小値。
WEOF	*end-of-file* の指定に使用されます。

宣言されている型

wchar ヘッダーでは、以下の型が宣言されています。

宣言されている型	説明
wchar_t	int 型 - 全拡張文字セット用コードを保持可能です。
size_t	unsigned int 型。
wint_t	int 型 - 拡張セットのメンバに該当する値を保持します。
mbstate_t	オブジェクト型 - 変換状態情報を保持します。
struct tm	不完全構造体型 - カレンダー時間のコンポーネントを保持します。

移植性

Ch の現行リリースでは、次の関数はサポートされていません。**fwprintf**()、**fwscanf**()、**swprintf**()、**swscanf**()、**vfwprintf**()、**vfwscanf**()、**vswprintf**()、**vswscanf**()、**vwprintf**()、**vwscanf**()、**wprintf**()、**wscanf**()。

_wfopen

Synopsis
#include <**stdio.h**>
file *_**wfopen**(**const** wchar_t **filename*, **const** wchar_t **mode*);

Purpose
Open a file.

Return Value
The _**wfopen** function returns a pointer to the object controlling the stream. If the open operation fails, _**wfopen** retruns a null pointer.

Parameters
filename Pointer to a file.

mode Points to a string.

Description
The _**wfopen** function opens the file whose name is the string pointed to by *filename*, and associates a stream with it.

The argument *mode* points to a string. If the string is one of the following, the file is open in the indicated mode. Otherwise, the behavior is undefined.

The arguments of _**wfopen**() are wide-character strings. The functions _**wfopen**() and **fopen**() behave identical, otherwise.

Portability
The function _**wfopen**() is available in Windows only.

See Also
fopen().

btowc

Synopsis
#include <**wchar.h**>
#include <**stdio.h**>
wint_t btowc(int *c***);**

Purpose
Determines if character is multibyte character.

Return Value
The **btowc**() function returns **WEOF** if *c* has the value **EOF** or if **(unsigned char)** *c* does not constitute a valid (one-byte) multibyte character in the initial shift state. Otherwise, it returns the wide-character representation of the character.

Parameters
c Integer argument.

Description
The **btowc**() function determines whether *c* constitutes a valid (one-byte) multibyte character in the initial shift state.

Example

```
/*  This is an example of btowc(int).  The function determines whether c is a
    valid (one-byte) character in the initial shift state.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    int c;
    wint_t retval;
    c='r';

    retval = btowc(c);
    if(retval != WEOF) {
     printf("btowc() = ");
     fputwc(c, stdout);
     printf("\n");
    }
    else
        printf("Error\n");
}
```

Output

```
btowc() = r
```

See Also

fgetwc

Synopsis
#include <**wchar.h**>
#include <**stdio.h**>
wint_t fgetwc(FILE **stream***);*

Purpose
Obtains wide character and advances one position.

Return Value
The **fgetwc**() function returns the next wide character from the input stream pointed to by *stream*. If the stream is at end-of-file, the end-of-file indicator for the stream is set and **fgetwc**() returns **WEOF**. If a read error occurs, the error indicator for the stream is set and **fgetwc**() returns **WEOF**. If an encoding error occurs (including too few bytes), the value of the macro **EILSEQ** is stored in **errno**() and **fgetwc**() returns **WEOF**.

Parameters
stream FILE pointer argument.

Description
If a next wide character is present from the input stream pointed to by *stream*, the **fgetwc**() function obtains that wide character and advances the associated file position indicator for the stream (if defined).

Example

```
/*This is an example of how to use the function fgetwc(FILE *).  It will
  retrieve a wide character from an input stream.
*/
#include <wchar.h>

int main() {
    FILE *stream;

    printf("Displays the first character from the file, ""wctype.c""\n");
    if((stream = fopen("fgetwc.c", "r")) != NULL) {
        printf("fgetwc(stream) = ");
        fputwc( fgetwc(stream), stdout);
        printf("\n");
    }
    else
        printf("Error in opening file!\n");
    fclose(stream);
}
```

Output

```
Displays the first character from the file, wctype.c
fgetwc(stream) = /
```

See Also

fgetws

Synopsis
#include <wchar.h>
#include <stdio.h>
wchar_t fgetws(wchar_t * restrict *s***, int** *n***, FILE * restrict** *stream***);**

Purpose
Reads wide characters from stream.

Return Value
The **fgetws**() function returns *s* if successful. If end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read or encoding error occurs during the operation, the array contents are indeterminate and a null pointer is returned.

Parameters
s Integer pointer argument.
n Integer argument.
stream FILE pointer argument.

Description
The **fgetws**() function reads at most one less than the number of wide characters specified by *n* from the stream pointed to by *stream* into the array pointed to by *s*. No additional wide characters are read after a new-line wide character (which is retained) or after end-of-file. A null wide character is written immediately after the last wide character read into the array.

Example

```
/* This is an example of the function fgetws(wchar_t*, int, FILE *).
   This will retrieve an entire string of the max length specifed by the user
   from an input stream.
*/
#include <wchar.h>

int main() {
    FILE *stream;
    int i;
    wchar_t str[100];

    i =10;
    printf("Retrieving a string of length %d from \"Makefile\".\n", i);
    if((stream = fopen("Makefile", "r")) != NULL) {
     fgetws(str, i, stream);
        fputws(str, stdout);
        fputs("\n", stdout);
    }
```

```
    else
        printf("Error in opening file!\n");
    fclose(stream);
}
```

Output

```
Retrieving a string of length 10 from "Makefile".
default a
```

See Also

fputwc

Synopsis
#include <**wchar.h**>
#include <**stdio.h**>
wint_t fputwc(wchar_t *c***, FILE** **stream***);*

Purpose
Writes the wide character to the output stream.

Return Value
The **fputwc**() function returns the next wide character written. If a write error occurs, the error indicator for the stream is set and **fputwc**() returns **WEOF**. If an encoding error occurs, the value of the macro **EILSEQ** is stored in **errno**() and **fputwc**() returns **WEOF**.

Parameters
c Integer argument.
stream FILE pointer argument.

Description
The **fputwc**() function writes the wide character specified by *c* to the output stream pointed to by *stream*, at the position indicated by the associated file position indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the character is appended to the output stream.

Example

```
/* This is an example of the function fputwc(wint_t, FILE *).  This will insert
   a wide character into the specifed output destination.
*/
#include <wchar.h>

int main() {
    FILE *stream;
    wint_t wc;
    char *name;

    if((name = tmpnam(NULL)) == NULL)
    printf("Error in creating temp file.\n");
    else {
wc = 'i';
     printf("Will write the character 'i' to to a temporary file.\n");
     if((stream = fopen(name, "w")) != NULL) {
            printf("fputwc(stream) = ");
            fputwc(wc,stream);
      fputwc(wc,stdout);
            printf("\n");
```

```
        }
      else
    printf("Error in opening file!\n");
    }
    fclose(stream);
    remove(name);
}
```

Output

```
Will write the character 'i' to to a temporary file.
fputwc(stream) = i
```

See Also

fputws

Synopsis
#include <**wchar.h**>
#include <**stdio.h**>
wchar_t fputws(const wchar_t * restrict *s***, FILE * restrict** *stream***);**

Purpose
Writes a wide string to stream.
Return Value
The **fputws**() function returns **EOF** if a write or encoding error occurs; otherwise, it returns a nonnegative value.

Parameters
s Integer pointer argument.
stream FILE pointer argument.

Description
The **fputws**() function writes the wide string pointed to by *s* to the stream pointed to by *stream*. The terminating null wide character is not written.

Example

```
/* This is an example for the function fputws(const wchar_t *, FILE *).
   It will place a wide character string into the specifed file.
   If the file does not exist, one will be created.
*/
#include <wchar.h>

int main() {
    FILE *stream;
    wchar_t line[100];
    char *temp, *name;
    int len;

    if((name = tmpnam(NULL)) == NULL)
     printf("temp file name could not be made. \n");
    else {
     temp ="Have a Nice Day!";
     len = strlen(temp);
     mbstowcs(line, temp, len);
     if((stream = fopen(name, "w")) != NULL) {
            fputws(line, stream);
            fputws(line, stdout);
            fputs("\n", stdout);
     }
        else
            printf("Error in opening file!\n");
```

```
    }
    fclose(stream);
    remove(name);
}
```

Output

```
Have a Nice Day!
```

See Also

fwide

Synopsis
#include <**wchar.h**>
#include <**stdio.h**>
int fwide(FILE **stream***,* **int** *mode***);**

Purpose
Determines orientation of stream.

Return Value
The **fwide**() function returns a value greater than zero if, after the call, the stream has wide orientation, a value less than zero if the stream has byte orientation, or zero if the stream has no orientation.

Parameters
stream FILE pointer argument.
mode Integer argument.

Description
The **fwide**() function determines the orientation of the stream pointed to by *stream*. If *mode* is greater than zero, the function first attempts to make the stream wide oriented. If *mode* is less than zero, the function first attemps to make the stream byte oriented. Otherwise, *mode* is zero and the function does not alter the orientation of the stream.

Example

```
/*  This is an example of the function fwide(FILE *, int).  The function will
    alter the stream according to the desired mode.
*/
#include <wchar.h>

int main() {
    FILE *stream;
    int c,retval;

    c = 11;
    if((stream = fopen("fwide.c", "r")) != NULL)
    {
        retval = fwide(stream, c);
        if(retval < 0)
            printf("Stream has byte orientation.\n");
        else if(retval > 0)
            printf("Stream has wide orientation.\n");
        else
            printf("Stream has no orientation.\n");
    }
    else
```

```
        printf("Error in opening file!\n");
    fclose(stream);
}
```

Output

```
Stream has wide orientation.
```

See Also

getwc

Synopsis
#include <**wchar.h**>
#include <**stdio.h**>
wint_t getwc(FILE *stream);

Purpose
Gets a wide character and advances the pointer.

Return Value
The **getwc**() function returns the next wide character from the input stream pointed to by *stream*, or **WEOF**.

Parameters
stream FILE pointer argument.

Description
The **getwc**() function is equivalent to **fgetwc**(), except that if it is implemented as a macro, it may evaluate *stream* more than once, so the argument should never be an expression with side effects.

Example

```
/* This is an example for the function getwc(FILE *).  This retrieves a wide
   character from the input stream.
*/
#include <wchar.h>

int main() {
    FILE *stream;
    int i;
    wint_t ch;

    printf("Retrieves the first 10 characters from the file ""wctype.c"".\n");
    if((stream = fopen("getwc.c", "r")) != NULL) {
  for(i =0; i<10; i++) {
          ch = getwc(stream);
         fputwc(ch, stdout);
     }
        printf("\n");
    }
    else
        printf("Error in opening file!\n");
    fclose(stream);
}
```

Output

```
Retrieves the first 10 characters from the file wctype.c.
/* This is
```

See Also

getwchar

Synopsis
#include <**wchar.h**>
wint_t getwchar(*void***);**

Purpose
Gets a wide character and advances the file position indicator.

Return Value
The **getwchar**() function returns the next wide character from the input stream pointed to by **stdin**, or **WEOF**.

Parameters
void Void argument.

Description
The **getwchar**() function is equivalent to **getwc** with the argument **stdin**.

Example

```
/* This is an example for getwchar(void).  This function will retrieve a wide
   character from the input stream, such as from stdin.  This program is
   executed by: getwchar.c < getwchar.in
   The input file getwchar.in contains: 'i'
*/
#include <wchar.h>

int main() {
    int i, ch;
    char buffer[80];

    printf("Enter a character. \n");
    for(i =0; (i<80) && (ch !='\n'); i++) {
     ch = getwchar();
        buffer[i]= (char)ch;

    }
    printf("You entered: %s\n",buffer);
}
```

Output

```
Enter a character.
You entered: i
```

See Also

mbrlen

Synopsis
#include <wchar.h>
size_t mbrlen(const char * restrict *s***, size_t** *n***, mbstate_t * restrict** *ps***);**

Purpose
Determines the number of bytes in a multibyte character.

Return Value
The **mbrlen**() function returns a value between zero and *n*, inclusive, $(\mathbf{size\_t}) - \mathbf{2}$, or $(\mathbf{size\_t}) - \mathbf{1}$.

Parameters
s Integer pointer argument.
n Unsigned integer argument.
ps Object type pointer argument.

Description
The **mbrlen**() function is equivalent to the call:

$$\mathbf{mbrtowc}(\mathbf{NULL}, s, n, ps! = \mathbf{NULL}?ps : \&\mathbf{internal})$$

where **internal** is the **mbstate_t** object for the **mbrlen**() function, except that the expression designated by *ps* is evaluated only once.

Example

```
/* This is an example of the function mbrlen(const char*, size_t, mbstate_t).
   This returns a value between zero and length.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    char *str;
    size_t retval, length;
    mbstate_t *ps;

    str = "hello";
    ps = NULL;
    length = 5;
    retval = mbrlen(str, length, ps);
    if(retval != (size_t) -1) {
     printf("Number of bytes of next character:  %d\n", retval);
    }
    else
        printf("Error\n");
```

```
}
```

Output

```
Number of bytes of next character:  1
```

See Also
mbrtowc()

mbrtowc

Synopsis
#include <wchar.h>
size_t mbrtowc(const char * restrict *pwc***, const char * restrict** *s***, size_t** *n***, mbstate_t * restrict** *ps***);**

Purpose
Determines the number of bytes needed to complete the next multibyte character.

Return Value
The **mbrtowc**() function returns the first of the following that applies (given the current conversion state)

0 if the next *n* or fewer bytes complete the multibyte character that corresponds to the null wide character (which is the value stored).

positive if the next *n* or fewer bytes complete a valid multibyte character (which is the value stored); the value returned is the number of bytes that complete the multibyte character.

$(\mathbf{size\_t}) - \mathbf{2}$ if the next *n* bytes contribute to an incomplete (but potentially valid) multibyte character, and all *n* bytes have been processed (no value is stored).

$(\mathbf{size\_t}) - \mathbf{1}$ if an encoding error occurs, in which case the next *n* or fewer bytes do not contribute to a complete and valid multibyte character (no value is stored); the value of the macro **EILSEQ** is stored in **errno**(), and the conversion state is undefined.

Parameters
pwc Character pointer argument.
s Character pointer argument.
n Unsigned integer argument.
ps Integer argument.

Description
If *s* is a null pointer, the **mbrtowc**() function is equivalent to the call:

mbrtowc(NULL, “”, 1, *ps***)**

In this case, the values of the parameters *pwc* and *n* are ignored.

If *s* is not a null pointer, the **mbrtowc**() function inspects at most *n* bytes beginning with the byte pointed to by *s* to determine the number of bytes needed to complete the next multibyte character (including any shift sequences). If the function determines that the next multibyte character is completed, it determines

the value of the corresponding wide character and then, if *pwc* is not a null pointer, stores that value in the object pointed to by *pwc*. If the corresponding wide character is the ull wide character, the resulting state described is the initial conversion state.

Example

```
/* This is an example of the function mbrtowc(const char*, size_t, mbstate_t).
   the mbrtowc function inspects at most n bytes beginning with the byte
   pointed  to by s to determine the number of bytes needed to complete the
   next multibyte character
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    char *str2;
    wchar_t *str;
    size_t retval, length;
    mbstate_t *ps;

    str2 = "hello";
    ps = NULL;
    length = 5;
    retval = mbrtowc(str, str2, length, ps);
    if(retval != (size_t) -1)
     printf("Number of bytes of the next character:  %d\n", retval);
    else
        printf("error\n");
}
```

Output

```
Number of bytes of the next character:  1
```

See Also

mbrtowc

mbsinit

Synopsis
#include <**wchar.h**>
#include <**stdio.h**>
int mbsinit(const mbstate_t ******ps***);**

Purpose
Determines if pointed-to object describes an initial conversion state.

Return Value
The **mbsinit**() function returns nonzero if *ps* is a null pointer or if the pointed-to object describesan initial state; otherwise it returns zero.

Parameters
ps Integer pointer argument.

Description
If *ps* is not a null pointer, the **mbsinit**() function determines whether the pointed-to **mbstate_t** object describes an initial conversion state.

Example

```
/* This is an example of the function mbsinit(mbstate_t).
   This function determines whether the object pointed to by ps describes an
   initial conversion state.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    int retval;
    mbstate_t *ps;

    ps = NULL;
    retval = mbsinit(ps);
    if(retval != 0) {
     printf("mbsinit(ps) returns nonzero if ps is a null pointer \n");
     printf("or ps is pointing to an initial conversion state.\n");
     printf("mbsinit(ps)= %d\n", retval);
    }
    else
        printf("Error \n");
}
```

Output

```
mbsinit(ps) returns nonzero if ps is a null pointer
```

```
or ps is pointing to an initial conversion state.
mbsinit(ps)= 1
```

See Also

mbsrtowcs

Synopsis
#include <**wchar.h**>
size_t mbsrtowcs(wchar_t * restrict *dst***, const char ** restrict** *src***, size_t** *len***, mbstate_t * restrict** *ps***);**

Purpose
Converts a sequence of multibyte charaters from an array into a sequence.

Return Value
If the input conversion encounters a sequence of bytes that do not form a valid multibyte character, an encoding error occurs: the **mbsrtowcs**() function stores the value of the macro **EILSEQ** in **errno**() and returns (**size_t**) **- 1**; the conversion state is undefined. Otherwise, it returns the number of multibyte characters successfully converted, not including the terminating null (if any).

Parameters
dst Character pointer argument.
src Character pointer argument.
len Unsigned integer argument.
ps Object type pointer.

Description
The **mbsrtowcs**() function converts a sequence of multibyte characters, beginning in the conversion state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a sequence of corresponding wide characters. If *dst* is not a null pointer, the converted characters are stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null character, which is also stored. Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multibyte character, or (if *dst* is not a null pointer) when *len* codes have been stored into the array pointed to by *dst* Each conversion takes place as if by call to the **mbrtowc** function.

If *dst* is not a null pointer, the pointer object pointed to by *src* is assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multibyte character converted (if any). If conversion stopped due to reaching a terminating null character and if *dst* is not a null pointer, the resulting state described is the initial conversion state.

Example

```
/* This is an example of the function mbsrtowcs(wchar_t*, const char**,
                        size_t, mbstate_t). Converts a sequence of characters
   into a sequence of corresponding wide-characters.
*/
#include <wchar.h>
#include <stdlib.h>
```

```
int main() {
    wchar_t str1[100];
    const char *str2;
    size_t retval, length;
    mbstate_t *ps;

    str2 = "hello";
    ps = NULL;
    length = 5;
    retval = mbsrtowcs(str1, &str2,length, ps);
    if(retval != (size_t) -1) {
     printf("Number of multibye characters");
     printf(" successfully converted: %d\n",retval);
    }
    else
        printf("Error\n");
}
```

Output

```
Number of multibye characters successfully converted: 5
```

See Also

putwc

Synopsis
#include <**wchar.h**>
#include <**stdio.h**>
wint_t putwc(wchar_t *c***, FILE** **stream***);*

Purpose
Writes a wide character to an output stream.

Return Value
The **putwc**() function returns the next wide character written, or **WEOF**.

Parameters
c Integer argument.
stream FILE pointer argument.

Description
The **putwc**() function is equivalent to **fputwc**(), except that if it is implemented as a macro, it may evaluate *stream* more than once, so that argument should never be an expression with side effects.

Example

```
/* This is an example for putwc(wint_t, FILE*).  This function will write a
  character to a stream.
*/
#include <wchar.h>

int main() {
    FILE *stream;
    wint_t wc;
    char *name;

    wc = 'i';
    if((name = tmpnam(NULL)) == NULL)
         printf("Error in creating temp file.\n");
    else {
      fputs("Will write the letter '",stdout);
     fputwc(wc, stdout);
     fputs("' to a temporary file.\n", stdout);
     if((stream = fopen(name, "w")) != NULL) {
            printf("putwc(stream) = %c\n", putwc(wc,stream));
        }
     else
            printf("Error in opening file!\n");
    }
    fclose(stream);
    remove(name);
```

```
}
```

Output

```
Will write the letter 'i' to a temporary file.
putwc(stream) = i
```

See Also

putwchar

Synopsis
#include <**wchar.h**>
wint_t putwchar(wchar_t *c***);**

Purpose
Writes a wide character to an output stream.

Return Value
The **putwchar**() function returns the character written, or **WEOF**.

Parameters
c Integer argument.

Description
The **putwchar**() function is equivalent to **putwc**() with the second argument **stdout**.

Example

```
/* This is an example of the function fputwchar(wint_t).  This will write a
   wide character to stdout.
*/
#include <wchar.h>

int main() {
    wint_t wc;
    wc = 's';

    printf("Will place the character '");
    fputwc(wc, stdout);
    printf("' on to stdout.\n");
    printf("putwchar(wc) = ");
    putwchar(wc);
    printf("\n");
    }
}
```

Output

```
Will place the character 's' on to stdout.
putwchar(wc) = s
```

See Also

ungetwc

Synopsis
#include <**wchar.h**>
#include <**stdio.h**>
wint_t ungetwc(wint_t *c***, FILE** **stream***);*

Purpose
Pushes a wide character back onto an input stream.

Return Value
The **ungetwc**() function returns the character pushed back, or **WEOF** if the operation fails.

Parameters
c Integer argument.
stream FILE pointer argument.

Description
The **ungetwc**() function pushes the wide character specified by *c* back onto the input stream pointed to by *stream*. Pushed-back wide characters will be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by *stream*) to a file positioning function (**fseek**(), **fsetpos**(), or **rewind**()) discards any pushed-back wide characters for the stream. The external storage corresponding to the stream is unchanged.

One wide character of pushback is guaranteed, even if the call to the **ungetwc**() function follows just after a call to a formatted wide character input function **fwscanf**(), **vfwscanf**(), **vwscanf**(), or **wscanf**(). If the **ungetwc**() function is called too many times on the same stream without an intervening read or file positioning operation on that stream, the operation may fail.

If the value of *c* equals that of the macro **WEOF**, the operation fails and the input stream is unchanged.

A successful call to the **ungetwc**() function clears the end-of-file indicator for the stream. The value of the file position indicator for the stream after reading or discarding all pushed-back wide characters is the same as it was before the wide characters were pushed back. For a text or binary stream, the value of its file position indicator after a successful call to the **ungetwc**() function is unspecified until all pushed-back wide characters are read or discarded.

Example

```
/* This is an example of the function ungetwc(wint_t, FILE *).  This pushes
   a character back onto the stream.
*/
```

```
#include <wchar.h>

int main() {
    FILE *stream;
    wint_t wc;

    if((stream = fopen("ungetwc.c", "r")) != NULL) {
        wc = fgetwc(stream);
        printf("ungetwc(wc,stream) = ");
        fputwc( ungetwc(wc,stream), stdout);
        printf("\n");
    }
    else
        printf("Error in opening file!\n");
    fclose(stream);
}
```

Output

```
ungetwc(wc,stream) = /
```

See Also

wcrtomb

Synopsis
#include <**wchar.h**>
size_t wcrtomb(const char * restrict *s***, wchar_t** *n***, mbstate_t * restrict** *ps***);**

Purpose
Determines the number of bytes needed to represent the next multibyte character.

Return Value
The **wcrtomb**() function returns the number of bytes stored in the array object (including any shift sequences). When *wc* in not a valid wide character, an encoding error occurs: the function stores the value of the macro **EILSEQ** in **errno**() and returns $(size\_t)$ **- 1**; the conversion state is undefined.

Parameters
s Character pointer argument.
n Character argument.
ps Integer pointer argument.

Description
If *s* is a null pointer, the **wcrtomb**() fucntion is equivalent to the call:

$$\mathbf{wcrtomb}(\mathbf{buf}, \mathbf{L'\backslash 0'}, \mathbf{1}, ps)$$

where **buf** is an internal buffer.

If *s* is not a null pointer, the **wcrtomb**() function determines the number of bytes needed to represent the multibyte character given by *wc* (including any shift sequences), and stores the resulting bytes in the array whose first element is pointed to by *s*. At most $\mathbf{MB\_CUR\_MAX}$ bytes are stored. If *wc* is a null wide character, a null byte is stored, preceded by any shift sequence needed to restore the initial shift state; the resulting state described is the initial conversion state.

Example

```
/* This is an example of the function wcrtomb(char*, wchar_t, mbstate_t).
   The wcrtomb function determines the number  of  bytes  needed  to represent
   the multibyte  character  that corresponds to the wide character given by wc.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    char *str;
    size_t retval;
```

```
    wchar_t c;
    mbstate_t *ps;

    str = "hello";
    c = 'e';
    ps = NULL;
    retval = wcrtomb(str, c, ps);
    if(retval != (size_t) -1) {
     printf("Number of bytes needed to represent the \n");
     printf("next character:  %d\n", retval);
    }
    else
        printf("Error\n");
}
```

Output

```
Number of bytes needed to represent the
next character:  1
```

See Also

wcscat

Synopsis
#include <**wchar.h**>
wchar_t wcscat(wchar_t * restrict *s1***, const wchar_t * restrict s2);**

Purpose
Appends a copy of one wide string to another wide string.

Return Value
The **wcscat**() returns the value of *s1*.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.

Description
The **wcscat**() function appends a copy of the wide string pointed to by *s2* (including the terminating null wide character) to the end of the wide string pointed to by *s1*. The initial wide character of *s2* overwrites the null character at the end of *s1*.

Example

```
/* This is an example of the function wcscat(wchar_t*, const wchar_t*).  This
   appends a string with another wide character string.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wchar_t str1[100];
    wchar_t str2[100];
    char *line;
    int retval, len;
    line = "Hello";
    len = strlen(line);

    mbstowcs(str1, line, len );
    mbstowcs(str2, line, len);
    fputs("String1: ", stdout);
    fputws(str1, stdout);
    fputs("\nString2: ", stdout);
    fputws(str2, stdout);
    wcscat(str1, str2);
    fputs("\nAfter wide character appending: ", stdout);
    fputws(str1, stdout);
    fputs("\n", stdout);
}
```

Output

```
String1: Hello
String2: Hello
After wide character appending: HelloHello
```

See Also

wcschr

Synopsis
#include <wchar.h>
wchar_t wcschr(const wchar_t *s, wchar_t c);

Purpose
Locates first occurance of a wide character in a string.

Description
The **wcschr**() function locates the first occurance of *c* in the wide string pointed to by *s*. The terminating null wide character is considered to be part of the wide string.

Parameters
s Integer pointer argument.
c Integer argument.

Return Value
The **wcschr**() function returns a pointer to the located wide string character, or a null pointer if the wide character does not occur in the wide string.

Example

```
/* This is an example of the function wcschr(const wchar_t*, wchar_t*).
   This function finds a character in a string.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wchar_t str[100], c;
    char *line;
    wchar_t *retval;
    int result, len;
    line = "Hello";
    c = 'l';

    len = strlen(line);
    mbstowcs(str, line, len);
    retval = wcschr(str,c);
    result = retval - str +1;
    if(result <= 0) {
  printf("There is no '");
        fputwc(c, stdout);
        printf("' in this string.\n");
    }
    else {
     printf("The first occurence of '");
```

```
        fputwc(c,stdout);
        printf("’ is at position: %d\n", result);
    }
}
```

Output

```
The first occurence of ’l’ is at position: 3
```

See Also

wcscmp

Synopsis
#include <**wchar.h**>
int wcscmp(const wchar_t **s1**, **const wchar_t** **s2**, **size_t** *n***);**

Purpose
Compares two wide strings.

Return Value
The **wcscmp**() function returns an integer greater than, equal to, or less than zero, accordingly as the wide string pointed to by *s1* is greater than, equal to, or less than the wide string pointed to by *s2*.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.
n Unsigned integer argument.

Description
The **wcscmp**() function compares the wide string pointed to by *s1* to the wide string pointed to by *s2*.

Example

```
/* This is an example of the function wcscmp(const wchar_t*, const wchar_t*).
   This function compares two strings.  If string1 < string2, then a negative
   number will be returned.  If string1 = string2, zero will be returned.
   If string1 > string2, a positive number will be returned.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    const char *line, *line2;
    wchar_t str1[100], str2[100];
    int retval,len1, len2;

    line = "hello";
    line2 = "hello";
    len1 = strlen(line);
    len2 = strlen(line2);
    mbstowcs(str1, line, len1);
    mbstowcs(str2, line2, len2);
    retval = wcscmp(str1, str2);
    if(retval > 0)
        printf("String1 is greater than String2\n");
    else if(retval == 0)
        printf("String1 is identical to String2\n");
    else if(retval < 0)
```

```
     printf("String1 is less than String2\n");
    else
        printf("Error!\n");
}
```

Output

```
String1 is identical to String2
```

See Also

wcscoll

Synopsis
#include <**wchar.h**>
int wcscoll(const wchar_t **s1***,* **const wchar_t** **s2***);*

Purpose
Compares two wide strings.

Return Value
The **wcscoll**() function returns an integer greater than, equal to, or less than zero, accordingly as the wide string pointed to by *s1* is greater than, equal to, or less than the wide string pointed to by *s2* when both are interpreted as appropriate to the current locale.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.

Description
The **wcscoll**() function compares the wide string pointed to by *s1* to the wide string pointed to by *s2*, both interpreted as appropriate to the **LC_COLLATE** category of the current locale.

Example

```
/* This is an example of the function wcscoll(const wchar_t*, const wchar_t*).
   If string1 < string2, a negative number will be returned.
   If string1 = string2, zero will be returned.  If
   string1 > string2, a positive number will be returned.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    const char *line, *line2;
    wchar_t str1[100], str2[100];
    int retval,len1,len2;

    line = "hello";
    line2 = "hello";
    len1 = strlen(line);
    len2 = strlen(line2);
    mbstowcs(str1, line, len1);
    mbstowcs(str2, line2, len2);
    retval = wcscoll(str1, str2);
    if(retval > 0)
        printf("String1 is greater than String2\n");
    else if(retval == 0)
        printf("String1 is identical to String2\n");
```

```
    else if(retval < 0)
        printf("String1 is less than String2\n");
    else
        printf("Error!\n");
}
```

Output

```
String1 is identical to String2
```

See Also

wcscpy

Synopsis
#include <wchar.h>
wchar_t wcscpy(wchar_t * restrict *s1***, const wchar_t * restrict** *s2***);**

Purpose
Copies a wide string to an array.

Return Value
The **wcscpy**() returns the value of *s1*.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.

Description
The **wcscpy**() function copies the wide string pointed to by *s2* (including the terminating null wide character) into the array pointed to by *s1*.

Example

```
/* This is an example of the function wcscpy(wchar_t*, const wchar_t*).  This
   make a copy of the second string and store into the first parameter.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wchar_t str1[100], str2[100];
    char *line;
    int retval,len;
    line = "Hello";
    len = strlen(line);

    mbstowcs(str2, line, len);
    wcscpy(str1, str2);
    fputws(str1, stdout);
    fputs("\n",stdout);
}
```

Output

```
Hello
```

See Also

wcscspn

Synopsis
#include <**wchar.h**>
size_t wcscspn(const wchar_t **s1***, const wchar_t** **s2***);**

Purpose
Computes the length of the maximum initial segment of a wide string.

Return Value
The **wcscspn**() function returns the length of the segment.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.

Description
The **wcscspn**() function computes the length of the maximum initial segment of the wide string pointed to by *s1* which consists entirely of wide characters *not* from the wide string pointed to by *s2*.

Example

```
/* This is an example of the function wcscspn(const wchar_t*, const wchar_t*).
   This finds a substring.  The function returns the length of the first
   substring that is made up of characters from string 2.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    char *line, *line2;
    wchar_t str1[100], str2[100];
    size_t retval;
    int len1,len2;
    line = "hello";
    line2 = "ll";

    len1 = strlen(line);
    len2 = strlen(line2);
    mbstowcs(str1, line,len1);
    mbstowcs(str2, line2, len2);
    retval = wcscspn(str1, str2);
    printf("Substring starts at position:  %d\n", retval);
}
```

Output

```
Substring starts at position:  2
```

See Also

wcsftime

Synopsis
#include <**wchar.h**>
size_t wcsftime(wchar_t * restrict *s***, size_t** *maxsize***, const wchar_t * restrict** *format***, const struct tm * restrict** *timeptr***);**

Purpose
Places wide characters into an array.

Return Value
If the total number of resulting wide characters, including the terminating null wide character is not more than *maxsize*, the **wcsftime**() function returns the number of wide characters placed into the array pointed to by *s* not including the terminating null wide character. Otherwise, zero is returned and the contents of the array are indeterminate.

Parameters
s Integer pointer argument.
maxsize Unsigned integer argument.
format Integer pointer argument.
timeptr Struct argument.

Description
The **wcsftime**() function is equivalent to the **strftime**() function, except that:

— The argument *s* points to the initial element of an array of wide characters into which the generated output is to be placed.

— The argument *maxsize* indicates the limiting number of wide characters into which the generated output is to be placed.

— The argument *format* is a wide string and the conversion specifiers are replaced by corresponding sequences of wide characters.

— The return value indicates the number of wide characters.

Example

```
/* This is an example of the function wcsftime(wchar_t*,size_t,const wchar_t*,
```

```
    const struct tm*).  This function formats a time string.
*/
#include <wchar.h>
#include <stdlib.h>
#include <time.h>

int main() {
    size_t retval, maxsize;
    wchar_t str1[100];
    wchar_t format[100];
    char *line;
    struct tm *timeptr;
    int len;
    time_t aclock;
    time(&aclock);
    maxsize = 100;

    line = "%c";
    len = strlen(line);
    mbstowcs(format, line, len);
    timeptr = localtime(&aclock);
    retval = wcsftime(str1, maxsize, format,timeptr );
    fputws(str1, stdout);
    fputs("\n", stdout);
}
```

Output

```
Mon Feb 21 14:56:31 2000
```

See Also

wcslen

Synopsis
#include <wchar.h>
size_t wcslen(const wchar_t *s);

Purpose
Computes the length of a wide string.

Return Value
The **wcslen**() function returns the number of wide characters that precede the terminating null wide character.

Parameters
s Integer pointer argument.

Description
The **wcslen**() function computes the length of the wide string pointed to by *s*.

Example

```
/* This is an example of the function wcslen(const wchar_t*).  This
   returns the length of the string.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    char *line;
    wchar_t str1[100];
    size_t retval;
    int len;

    line = "hello";
    len = strlen(line);
    mbstowcs(str1, line, len);
    retval = wcslen(str1);
    printf("Number of characters in str1:  %d\n", retval);
}
```

Output

```
Number of characters in str1:  5
```

See Also

wcsncat

Synopsis
#include <**wchar.h**>
wchar_t wcsncat(wchar_t * restrict *s1***, const wchar_t * restrict** *s2***, size_t** *n***);**

Purpose
Appends wide characters from an array to a wide string.

Return Value
The **wcsncat**() returns the value of *s1*.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.
n Unsigned integer argument.

Description
The **wcsncat**() function appends not more than *n* wide characters (a null wide character and those that follow it are not appended) from the array pointed to by *s2* to the end of the wide string pointed to by *s1*. The initial wide character of *s2* overwrites the null wide character at the end of *s1*. A terminating null wide character is always appended to the result.

Example

```
/* This is an example of the function wcsncat(wchar_t*, const wchar_t*, size_t).
    This function appends the specified number of characters from str2 to str1.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wchar_t str1[100], str2[100];
    char *line, *line2;
    wchar_t  *retval;
    size_t count;
    int len1,len2;

    line = "Hello, ";
    line2 ="how are you today?";
    len1 = strlen(line);
    len2 = strlen(line2);
    count =18;
    mbstowcs(str1, line, len1);
    mbstowcs(str2, line2, len2);
    wcsncat(str1,str2,count);
    fputws(str1, stdout);
    fputs("\n", stdout);
```

```
}
```

Output

```
Hello, how are you today?
```

See Also

wcsncmp

Synopsis
#include <wchar.h>
int wcsncmp(const wchar_t *s1, const wchar_t *s2, size_t n);

Purpose
Compares wide characters between two arrays.

Return Value
The **wcsncmp**() function returns an integer greater than, equal to, or less than zero, accordingly as the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the possibly null-terminated array pointed to by *s2*.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.
n Unsigned integer argument.

Description
The **wcsncmp**() function compares not more than *n* wide characters (those that follow a null wide character are not compared) from the array pointed to by *s1* to the array pointed to by *s2*.

Example

```
/* This is an example of the function wcsncmp(const wchar_t*, const wchar_t*,
   size_t).  This compares the specifed number of characters of two strings.
   If string1 substring < string2 substring, then a negative value is returned.
   If string1 substring = string2 substring, then zero is returned.
   If string1 substring > string2 substring, then a positive value is returned.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    char *line, *line2;
    wchar_t str1[100], str2[100];
    size_t count;
    int retval,len1,len2;
    count = 5;
    line = "hellohello";
    line2 = "hello";
    len1 = strlen(line);
    len2 = strlen(line2);

    mbstowcs(str1, line, len1);
    mbstowcs(str2, line2,len2);
    fputs("String1: ", stdout);
```

```
    fputws(str1, stdout);
    fputs("\nString2: ", stdout);
    fputws(str2, stdout);
    printf("\nNumber of characters to compare: %d\n", count);
    retval = wcsncmp(str1, str2, count);
    if(retval > 0)
        printf("String1 is greater than String2\n");
    else if(retval == 0)
        printf("String1 is identical to String2\n");
    else if(retval < 0)
        printf("String1 is less than String2\n");
    else
        printf("Error!\n");
    printf("wcsncmp(str1,str2,count) = %d\n", retval);
}
```

Output

```
String1: hellohello
String2: hello
Number of characters to compare: 5
String1 is identical to String2
wcsncmp(str1,str2,count) = 0
```

See Also

wcsncpy

Synopsis
#include <**wchar.h**>
wchar_t wcsncpy(wchar_t * restrict *s1***, const wchar_t * restrict** *s2***, size_t** *n***);**

Purpose
Copies wide characters from one array to another array.

Return Value
The **wcsncpy**() returns the value of *s1*.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.
n Unsigned integer argument.

Description
The **wcsncpy**() function copies not more than *n* wide characters (those that follow a null wide character are not copied) from the array pointed to by *s2* to the array pointed to by *s1*.

If the array pointed to by *s2* is a wide string that is shorter than *n* wide characters, null wide characters are appended to the copy in the array pointed to by *s1*, until *n* wide characters in all have been written.

Example

```
/*  This is an example of the function wcsncpy(wchar_t*, const wchar_t*,
    size_t). This copies the specified number of characters from string2
    to string1.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wchar_t str1[100], str2[100];
    char *line;
    wchar_t  *retval;
    size_t count;
    int len;

    line ="How are you today?";
    count =18;
    len = strlen(line);
    mbstowcs(str2, line,len);
    wcsncpy(str1,str2,count);
    fputws(str1, stdout);
    fputs("\n", stdout);
```

```
}
```

Output

```
How are you today?
```

See Also

wcspbrk

Synopsis
#include <**wchar.h**>
wchar_t wcspbrk(const wchar_t ***s1*****, const wchar_t** ***s2*****);**

Purpose
Locates the first occurance of a wide character from one string in another string.

Return Value
The **wcspbrk**() function returns a pointer to the wide character in *s1*, or a null pointer if no wide character from *s2* occurs in *s1*.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.

Description
The **wcspbrk**() function locates the first occurrence in the wide string pointed to by *s1* of any wide character from the wide string pointed to by *s2*.

Example

```
/* This is an example of the function wcspbrk(const wchar_t*, const wchar_t).
   Returns a pointer to the first character in string1 that also belongs to
   string2.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wchar_t str1[100], str2[100];
    char *line, *line2;
    wchar_t  *retval;
    int len1, len2;
    line = "Today is a very fine day.\n";
    line2 ="abc";
    len1 = strlen(line);
    len2 = strlen(line2);

    mbstowcs(str1, line, len1);
    mbstowcs(str2, line2, len2);
    retval = wcspbrk(str1,str2);
    fputs("Original: ", stdout);
    fputws(str1, stdout);
    fputs("Characters to search for: ", stdout);
    fputws(str2, stdout);
    fputs("\nFirst occurrence of a character belonging to string2: ", stdout);
```

```
    fputws(retval, stdout);
    fputs("\n", stdout);
}
```

Output

```
Original: Today is a very fine day.
Characters to search for: abc
First occurrence of a character belonging to string2: ay is a very fine day.
```

See Also

wcsrchr

Synopsis
#include <wchar.h>
wchar_t wcsrchr(const wchar_t *s, wchar_t c);

Purpose
Locates the last occurance of a wide character in a wide string.

Return Value
The **wcsrchr**() function returns a pointer to the located wide character, or a null pointer if the wide character does not occur in the wide string.

Parameters
s Integer pointer argument.
c Integer argument.

Description
The **wcsrchr**() function locates the last occurance of *c* in the wide string pointed to by *s*. The terminating null wide character is considered to be part of the wide string.

Example

```
/* This is an example of the function wcsrchr(const char_t*, wchar_t*).
   This returns a pointer to the last occurence of the search character.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wchar_t str1[100];
    wchar_t c, *retval;
    char *line;
    int result, len;
    line = "Hello, how are you doing?";
    c = 'l';

    len = strlen(line);
    mbstowcs(str1, line, len);
    fputs("String to be searched: ", stdout);
    fputws(str1, stdout);
    printf("\nCharacter to be searched for: ");
    fputwc(c, stdout);
    printf("\n");
    retval = wcsrchr(str1,c);
    result = retval - str1 +1;
    if(result <= 0) {
  printf("There is no '");
```

```
        fputwc(c, stdout);
        printf("' in this string.\n");
    }
    else {
     printf("The last occurence of '");
        fputwc(c, stdout);
        printf("' is at position: %d\n", result);
    }
}
```

Output

```
String to be searched: Hello, how are you doing?
Character to be searched for: l
The last occurence of 'l' is at position: 4
```

See Also

wcsrtombs

Synopsis
#include <**wchar.h**>
size_t wcsrtombs(wchar_t * restrict *dst***, const wchar_t ** restrict** *src***, size_t** *len***, mbstate_t * restrict** *ps***);**

Purpose
Converts a sequence of wide characters from an array into a sequence of multibyte characters.

Return Value
If conversion stops because a code is reached that does not correspond to a valid multibyte character, an encoding error occurs: the **wcsrtombs**() function stores the value of the macro **EILSEQ** in **errno**() and returns $(size\_t)$ **- 1**; the conversion state is undefined. Otherwise, it returns the number of bytes in the resulting multibyte character sequence, not including the terminating null (if any).

Parameters
dst Character pointer argument.
src Character pointer argument.
len Unsigned integer argument.
ps Object type pointer.

Description
The **wcsrtombs**() function converts a sequence of wide characters from the array indirectly pointed to by *src* into a sequence of corresponding multibyte characters, beginning in the conversion state described by the object pointed to by *ps*. If *dst* is not a null pointer, the converted characters are then stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null character, which is also stored. Conversion stops earlier in two cases: when a code is reached that does not correspond to a valid multibyte character, or (if *dst* is not a null pointer) when the next multibyte character would exceed the limit of *len* total bytes to be stored into the array pointed to by *dst*. Each conversion takes place as if by call to the **wcrtomb**() function.

If *dst* is not a null pointer, the pointer object pointed to by *src* is assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null character, the resulting state described is the initial conversion state.

Example

```
/* This is an example of the function wcsrtombs(char*, const wchar_t**,
                        size_t, mbstate_t).
   The wcsrtombs function  converts  a  sequence  of  wide
   characters  from the array indirectly pointed to by src into
```

```
   a sequence of corresponding multibyte characters,  beginning
   in  the  conversion state described by the object pointed to
   by ps.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    char *str1, *line;
    const wchar_t *str2;
    wchar_t str3[100];
    size_t retval, length;
    mbstate_t *ps;
    int len;

    line = "hello";
    len = strlen(line);
    mbstowcs(str3, line, len);
    str2 = str3;
    ps = NULL;
    length = 5;
    retval = wcsrtombs(str1, &str2,length, ps);
    if(retval != (size_t) -1) {
     printf("Number of multibye characters successfully");
        printf(" converted: %d\n",retval);
    }
    else
printf("Error\n");
}
```

Output

```
Number of multibye characters successfully converted: 5
```

See Also

wcsspn

Synopsis
#include <**wchar.h**>
size_t wcsspn(const wchar_t **s1***, const wchar_t** **s2***);**

Purpose
Computes the length of the maximum initial segment of a wide string.

Return Value
The **wcsspn**() function returns the length of the segment.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.

Description
The **wcsspn**() function computes the length of the maximum initial segment of the wide string pointed to by *s1* which consists entirely of wide characters from the wide string pointed to by *s2*.

Example

```
/* This is an example of the function wcsspn(const wchar_t*, const wchar_t*).
   This returns the length of the substring found in str1.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    char *line, *line2;
    wchar_t str1[100], str2[100];
    size_t retval;
    int len1, len2;

    line = "hello, how are you?";
    line2 = "hel";
    len1 = strlen(line);
    len2 = strlen(line2);
    mbstowcs(str1, line, len1);
    mbstowcs(str2, line2, len2);
    fputs("String: ", stdout);
    fputws(str1, stdout);
    fputs("\nSubstring to search for: ", stdout);
    fputws(str2, stdout);
    retval = wcsspn(str1, str2);
    printf("\nThe length of the substring is:  %d\n", retval);
}
```

Output

```
String: hello, how are you?
Substring to search for: hel
The length of the substring is:  4
```

See Also

wcsstr

Synopsis
#include <**wchar.h**>
wchar_t wcsstr(const wchar_t **s1*, **const wchar_t** **s2*);**

Purpose
Locates the first sequence found in one wide string in the other wide string.

Return Value
The **wcsstr**() function returns a pointer to the located wide string, or a null pointer if the wide string is not found. If *s2* points to a wide string with zero length, the function returns *s1*.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.

Description
The **wcsstr**() function locates the last occurance in the wide string pointed to by *s1* of the sequence of wide characters (excluding the terminating null wide character) in the wide string pointed to by *s2*.

Example

```
/*  This is the function wcsstr(const wchar_t*, const wchar_t*).
    This function finds the first occurence of a wide string from str2 in str1.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wchar_t str1[100], str2[100];
    wchar_t *retval;
    char *line, *line2;
    int len1, len2;
    line = "Today is a very fine day.";
    line2 ="is";
    len1 = strlen(line);
    len2 = strlen(line2);

    mbstowcs(str1, line, len1);
    mbstowcs(str2, line2, len2);
    retval = wcsstr(str1, str2);
    if(retval != NULL) {
     fputs("Original: ", stdout);
     fputws(str1, stdout);
     fputs("\nCharacters to search for: ", stdout);
     fputws(str2, stdout);
     printf("\nFirst occurrence of a character ");
```

```
        printf("belonging to string2: ");
        fputws(retval,stdout);
     fputs("\n", stdout);
    }
    else
        printf("Error\n");
}
```

Output

```
Original: Today is a very fine day.
Characters to search for: is
First occurrence of a character belonging to string2: is a very fine day.
```

See Also

wcstod

Synopsis
#include <wchar.h>
double wcstod(const wchar_t * restrict *nptr***, wchar_t ** restrict** *endptr***);**
float wcstof(const wchar_t * restrict *nptr***, wchar_t ** restrict** *endptr***);**
long double wcstold(const wchar_t * restrict *nptr***, wchar_t ** restrict** *endptr***);**

Purpose
Converts an initial portion of a wide string to double, float, and long double, respectively.

Return Value
The functions return the converted value, if any. If no conversions could be performed, zero is returned. If the correct value is outside the range of representable values, plus or minus **HUGE_VAL**, **HUGE_VALF**, or **HUGE_VALL** is returned (according to the return type and the sign of the value) and the value of the macro **ERANGE** is stored in **errno**(). If the result underflows, the functions return a value whose magnitude is no greater than the smallest normalized positive number in the return type; whether **errno**() acquires the value **ERANGE** is implementation defined.

Parameters
nptr Integer pointer argument.
endptr Integer pointer argument.

Description
The **wcstod**(), **wcstof**(), and **wcstold**() functions convert the initial portion of the wide string pointed to by *nptr* to **double**, **float**, and **long double** representation, respectively. First, they decompose the input string into three parts: an initial, possibly empty, sequence of white-space wide characters (as specified by the **iswspace**() function) a subject sequence resembling a floating-point constant or representing an infinity or NaN; and a final wide string of one or more unrecognized wide characters, including the terminating null wide character of the input wide string. Then, they attempt to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional plus or minus sign, then one of the following:

— a nonempty sequence of decimal digits optionally containing a decimal-point wide character, then an optional exponent part as defined for the corresponding single-byte characters;

— a **0x** or **0X**, then a nonempty sequence of decimal digits optionally containing a decimal-point wide character, then an optional binary exponent part;

— one of **INF** or **INFINITY**, or any other wide string equivalent.

— one of **NAN** or **NAN(**$n - wchar - sequence_{opt}$**)**, or any other wide string equivalent except for case in the NAN part, where:
n-wchar-sequence:
digit
nondigit
n-wchar-sequence digit
n-wchar-sequence nondigit

The subject sequence is defined as the longest initial subsequence of the input wide string, starting with the first non-white-space wide character, that is of the expected form. The subject sequence contains no wide characters if the input wide string is not of the expected form.

If the subject sequence has the expected form for a floating-point number, the sequence of wide characters starting with the first digit or the decimal-point wide character (whichever occurs first) is interpreted as a floating constant, except that the decimal-point wide character is used in place of a period, and that if neither an exponent part nor a decimal-point wide character appears in a decimal floating point number, or if a binary exponent part does not appear in a binary floating point number, an exponent part of the appropriate type with the value zero is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the sequence is interpreted as negated. A wide character sequence **INF** or **INFINITY** is interpreted as an infinity, if representable in the return type, else like a floating constant that is too large for the range of the return type. A wide character sequence **NAN** or **NAN(**$n - wchar - sequence_{opt}$**)** is interpreted as a quiet NaN, if supported in the return type, else like a subject sequence part that does not have the expected form; the meaning of the n-wchar sequences is implementation-defined. A pointer to the final wide string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

If the subject sequence has the hexadecimal form and **FLT_RADIX** is a power of 2, the value resulting from the conversion is correctly rounded.

In other than the **"C"** locale, additional locale-specific subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

Recommended practice If the subject sequence has the hexadecimal form and **FLT_RADIX** is not a power of 2, the result should be one of the two numbers in the appropriate internal format that are adjacent to the hexadecimal floating source value, with the extra stipulation that the error should have a correct sign for the current rounding direction.

If the subject sequence has the decimal form and at the most **DECIMAL_DIG** (defined in **float.h**) significant digits, the result should be correctly rounded. If the subject sequence *D* has the decimal form and more than **DECIMAL_DIG** significant digits, such that the values of *L* and *U*, both having **DECIMAL_DIG**

significant digits, such that the values of *L*, *D*, and *U* satisfy $L \leq D \leq U$. The result should be one of the (equal or adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current rounding direction, with the extra stipulation that the error with respect to *D* should have a correct sign for the current rounding direction.

Example

```
/* This is an example of the function wcstod(const wchar_t*, wchar_t**).
   This converts a strint to a double value.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wchar_t nptr[100];
    wchar_t *endptr;
    double x;
    char *line;
    int len;

    line = "3.1415926This stopped it";
    len = strlen(line);
    mbstowcs(nptr, line, len);
    x = wcstod(nptr, &endptr);
    printf("Original string: ");
    fputws(nptr, stdout);
    printf("\nDouble representation: %f\n", x);
}
```

Output

```
Original string: 3.1415926This stopped it
Double representation: 3.141593
```

See Also

wcstok

Synopsis
#include <**wchar.h**>
wchar_t wcstok(wchar_t * restrict *s1***, const wchar_t * restrict** *s2***, wchar_t ** restrict** *ptr***);**

Purpose
Breaks a wide string into a sequence of tokens.

Return Value
The **wcstok**() function returns a pointer to the first wide character of a token, or a null pointer if there is no token.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.
ptr Integer pointer argument.

Description
A sequence of calls to the **wcstok**() function breaks the wide string pointed to by *s1* into a sequence of tokens, each of which is delimited by a wide character from the wide string pointed to by *s2*. The third argument points to a caller-provided **wchar_t** pointer into which the **wcstok**() function stores information necessary for it to continue scanning the same wide string.

The first call in a sequence has a non-null first argument and stores an initial value in the object pointed to by *ptr*. Subsequent calls in the sequence have a null first argument and the object pointed to by *ptr* is required to have the value stored by the previous call in the sequence, which is then updated. The separator wide string pointed to by *s2* may be different from call to call.

The first call in the sequence searches the wide string pointed to by *s1* for the first wide character that is *not* contained in the current separator wide string pointed to by *s2*. If no such wide character is found, then there are no tokens in the wide string pointed to by *s1* and the **wcstok** function returns a null pointer. If such a wide character is found, it is the start of the first token.

The **wcstok**() function then searches from there for a wide character that *is* contained in the current separator wide string. If no such wide character is found, the current token extends to the end of the wide string pointed to by *s1*, and the subsequent searches in the same wide string for a token returns a null pointer. If such a wide character is found, it is overwritten by a null character, which terminates the current token.

In all cases, the **wcstok**() function stores sufficient information in the pointer pointed to by *ptr* so that

subsequent calls, with a null pointer for *s1* and the unmodified pointer value for *ptr*, shall start searching just past the element overwritten by a null wide character (if any).

Example

```
/* This is an example of the function wcstok(wchar_t*, const wchar_t*,
                                                wchar_t**).  This pulls out
   the tokens from a given string. There is a difference between the number
   of arguments depending on where you compile.  Portability difference
   between HP and Sun.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    char *line, *line2;
    wchar_t str1[100], str2[100];
    wchar_t *endptr = NULL;
    wchar_t *token;
    int len1, len2;

    line = "hello goodbye now";
    line2 = " ";
    len1 = strlen(line);
    len2 = strlen(line2);
    mbstowcs(str1, line, len1);
    mbstowcs(str2, line2, len2);
    fputs("Original string: ", stdout);
    fputws(str1, stdout); fputs("\n", stdout);
    for (token = wcstok(str1, str2, &endptr);
         token != NULL; token= wcstok(NULL, str2, &endptr)) {
      fputs("token = ", stdout); fputws(token, stdout); fputs("\n", stdout);
   }
}
```

Output

```
Original string: hello goodbye now
token = hello
token = goodbye
token = now
```

See Also

wcstol

Synopsis
#include <**wchar.h**>
long int wcstol(const wchar_t * restrict *nptr***, wchar_t ** restrict** *endptr***, int** *base***);**
long long int wcstoll(const wchar_t * restrict *nptr***, wchar_t ** restrict** *endptr***, int** *base***);**
unsigned long int wcstoul(const wchar_t * restrict *nptr***, wchar_t ** restrict** *endptr***, int** *base***);**
unsigned long long int wcstoull(const wchar_t * restrict *nptr***, wchar_t ** restrict** *endptr***, int** *base***);**

Purpose
Converts wide string to long int, long long int, unsigned long int, or unsigned long long int.

Return Value
The **wcstol**(), **wcstoll**(), **wcstoul**(), and **wcstoull**() functions return the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, **LONG_MIN**, **LONG_MAX**, **LLONG_MIN**, **LLONG_MAX**, **ULONG_MAX**, or **ULLONG_MAX** is returned (according to the return type sign of the value, if any) and the value of the macro **ERANGE** is stored in errno.

Parameters
nptr Integer pointer argument.
endptr Integer pointer argument.
base Integer argument.

Description
The **wcstol**(), **wcstoll**(), **wcstoul**(), and **wcstoull**() functions convert the initial portion of the wide string pointed to by *nptr* to **long int**, **long long int**, **unsigned long int**, and **unsigned long long int** representation, respectively. First, they decompose the input string into three parts: an initial, possibly empty, sequence of white-space wide characters (as specified by the **iswspace**() function) a subject sequence resembling an integer represented in some radix determined by the value of *base*, and a final wide string of one or more unrecognized wide characters, including the terminating null wide character of the input wide string. Then, they attempt to convert the subject sequence to an integer, and return the result.

If the value of *base* is zero, the expected form of the subject sequence is that of an integer constant as described for the corresponding single-byte characters, optionally preceded by a plus or minus sign, but not including an integer suffix. If the value of *base* is between 2 and 36 (inclusive) the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a plus or minus sign, but not including an integer suffix. The letters **a** (or **A**) through **z** (or **Z**) are ascribed the values of 10 through 35; only letters and digits whose ascribed values are less than that of *base* are permitted. If the value of *base* is 16, the wide characters **0x** or **0X** may optionally precede

the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input wide string, starting with the first non-white-space wide character, that is of the expected form. The subject sequence contains no wide characters if the input wide string is empty or consists entirely of white space, or if the first non-white-space wide character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is zero, the sequence of wide characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated (in the return type). A pointer to the final wide string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the **"C"** locale, additional locale-specific subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

Example

```
/* This is an example of the function wcstol(const wchar_t*,wchar_t** ,int).
    This converts a string to a long integer in the specified base.
*/
#include <wchar.h>

int main() {
    wchar_t nptr[100];
    wchar_t *endptr;
    int base, len;
    long retval;
    char *line;

    line= "-10110134932This stopped it";
    base = 10;
    len = strlen(line);
    mbstowcs(nptr, line, len);
    fputs("Original string: ", stdout);
    fputws(nptr, stdout);
    printf("\nLong integer representation in base %d : ", base);
    retval = wcstol(nptr, &endptr, base);
    printf("%ld\n", retval);
}
```

Output

```
Original string: -10110134932This stopped it
Long integer representation in base 10 : -2147483648
```

See Also

wcsxfrm

Synopsis
#include <**wchar.h**>
int wcsxfrm(const wchar_t * restrict *s1***, const wchar_t * restrict** *s2***, size_t** *n***);**

Purpose
Transforms a wide string and places it in an array.

Return Value
The **wcsxfrm**() function returns the length of the transformed wide string (not including the terminating null wide character). If the value returned in *n* or greater, the contents of the array pointed to by *s1* are indeterminate.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.
n Unsigned integer argument.

Description
The **wcsxfrm**() function transforms the wide string pointed to by *s2* and places the resulting wide string into the array pointed to by *s1*. The transformation is such that if the **wcscmp**() function is applied to two transformed wide strings, it returns a value greater than, equal to, or less than zero, corresponding to the result of the **wcscoll**() function applied to the same two original wide strings. No more than *n* wide characters are placed into the resulting array pointed to by *s1*, including the terminating null wide character. If *n* is permitted to be a null pointer.

Example

```
/* This is an example of the function wcsxfrm(wchar_t*, const wchar_t*,size_t).
   This trnasforms a string based on locale-specific information.  It returns
   the length of the transformed string.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    char *line, *line2;
    wchar_t str1[100], str2[100];
    size_t retval,count;
    int len1;

    line = "abc";
    len1 = strlen(line);
    mbstowcs(str2, line, len1);
    fputs("Source string: ", stdout);
```

```
    fputws(str2, stdout);
    retval = wcsxfrm(str1, str2,count);
    printf("\nThe length of the transformed string is:  %d\n", retval);
}
```

Output

```
Source string: abc
The length of the transformed string is:  3
```

See Also

wctob

Synopsis
#include <**wchar.h**>
#include <**stdio.h**>
int wctob(wint_t i*c*);

Purpose
Determines if a character corresponds to member of extended character set.

Description
The **wctob**() function determines whether *c* corresponds to a member of the extended character set whose multibyte character representation is a single byte when in the initial shift state.

Parameters
c Integer argument.

Return Value
The **wctob**() function returns **EOF** if *c* does not correspond to multibyte character with length one in the initial shift state. Otherwise, it returns the single-byte representation of that character as an **unsigned char** converted to an **int**.

Example

```
/*  This is an example of the function wctomb(char*, wchar_t).  This function
    determines the number of bytes needed to represent the equivalent mulitbyte
    character.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wint_t wc;
    int retval;
    wc=L'r';

    retval = wctob(wc);
    if(retval != EOF) {
        printf("wctob() = ");
      fputc(retval, stdout);
     printf("\n");
    }
    else
   printf("error\n");
}
```

Output

```
wctob() = r
```

See Also

wmemchr

Synopsis
#include <wchar.h>
wchar_t wmemchr(const wchar_t *s, wchar_t *c*, size_t *n*);

Purpose
Locates the first occurance of a specified wide character in a stream.

Return Value
The **wmemchr**() function returns a pointer to the located wide character, or a null pointer if the wide character does not occur in the object.

Parameters
s Integer pointer argument.
c Integer argument.
n Unsigned integer argument.

Description
The **wmemchr**() function locates the first occurance of *c* in the initial *n* wide characters of the object pointed to by *s*.

Example

```
/*  This is an example of wmemchr(const wchar_t*, wchar_t, size_t).
    This funtion finds and returns the located specified wide character 'c'.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    int len, result;
    size_t count;
    wchar_t *retval, str1[100], c;
    char *line;
    c='r';
    count = 20;

    line = "Hello how are you today?";
    len = strlen(line);
    mbstowcs(str1, line, len);
    retval = wmemchr(str1,c, count);
    if(retval != NULL) {
     printf("%s\n", line);
     result = retval - str1 +1;
     if(result <= 0)
            printf("There is no %c in this string.\n",c);
     else
```

```
            printf("The first ’%c’ is at position: %d\n", c, result);
    }
    else
        printf("Error\n");
}
```

Output

```
Hello how are you today?
The first ’r’ is at position: 12
```

See Also

wmemcmp

Synopsis
#include <wchar.h>
int wmemcmp(const wchar_t *s1, const wchar_t *s2, size_t n);

Purpose
Compares the first wide characters in two objects.

Return Value
The **wmemcmp**() function returns an integer greater than, equal to, or less than zero, accordingly as the object pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.
n Unsigned integer argument.

Description
The **wmemcmp**() function compares the first *n* wide characters of the object pointed to by *s1* to the first *n* wide characters of the object pointed to by *s2*.

Example

```
/* This is an example of the function wmemcmp(const wchar_t*, const wchar_t*,
   size_t).  This compares the specifed number of characters of two strings.
   If string1 substring < string2 substring, then a negative value is returned.
   If string1 substring = string2 substring, then zero is returned.
   If string1 substring > string2 substring, then a positive value is returned.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    const char *line, *line2;
    wchar_t str1[100], str2[100];
    size_t count;
    int retval,len1,len2;
    count = 6;
    line = "hellohello";
    line2 = "hello";
    len1 = strlen(line);
    len2 = strlen(line2);

    mbstowcs(str1, line, len1);
    mbstowcs(str2, line2,len2);
    retval = wmemcmp(str1, str2, count);
    if(retval != NULL) {
```

```
     fputs("String1: ", stdout);
     fputws(str1, stdout);
     fputs("\nString2: ", stdout);
     fputws(str2, stdout);
     printf("\nNumber of characters to compare: %d\n", count);
     if(retval > 0)
            printf("String1 is greater than String2\n");
     else if(retval == 0)
            printf("String1 is identical to String2\n");
     else if(retval < 0)
            printf("String1 is less than String2\n");
     else
            printf("Error!\n");
     printf("wmemcmp(str1,str2,count) = %d\n", retval);
    }
    else
        printf("Error\n");
}
```

Output

```
String1: hellohello
String2: hello
Number of characters to compare: 6
String1 is greater than String2
wmemcmp(str1,str2,count) = 104
```

See Also

wmemcpy

Synopsis
#include <**wchar.h**>
wchar_t wmemcpy(wchar_t * restrict *s1***, const wchar_t * restrict** *s2***, size_t** *n***);**

Purpose
Copies wide characters from one object to another.

Return Value
The **wmemcpy**() returns the value of *s1*.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.
n Unsigned integer argument.

Description
The **wmemcpy**() function copies *n* wide characters from the object pointed to by *s2* to the object pointed to by *s1*.

Example

```
/* This is an example of the function wmemcpy(wchar_t*, const wchar_t*,
                                                                  size_t).
   This copies the specified number of characters from string2 to string1.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wchar_t str1[100], str2[100];
    char *line;
    wchar_t  *retval;
    size_t count;
    int len;

    line ="How are you today?";
    count =11;
    len = strlen(line);
    mbstowcs(str2, line,len);
    retval = wmemcpy(str1,str2,count);
    if(retval != NULL) {
     printf("Original string: ");
     fputws(str2, stdout);
     printf("\nNumber of characters to copy: %d", count);
     printf("\nCopied string: ");
     fputws(str1, stdout);
```

```
     fputs("\n", stdout);
    }
    else
        printf("Error\n");
}
```

Output

```
Original string: How are you today?
Number of characters to copy: 11
Copied string: How are you
```

See Also

wmemmove

Synopsis
#include <**wchar.h**>
wchar_t wmemmove(wchar_t * restrict *s1***, const wchar_t * restrict** *s2***, size_t** *n***);**

Purpose
Copies wide characters from one object to another.

Return Value
The **wmemmove**() returns the value of *s1*.

Parameters
s1 Integer pointer argument.
s2 Integer pointer argument.
n Unsigned integer argument.

Description
The **wmemmove**() function copies *n* wide characters from the object pointed to by *s2* to the object pointed to by *s1*. Copying takes place as if the *n* wide characters from the object pointed to by *s2* are first copied into a temporary array of *n* wide characters that does not overlap the objects pointed to by *s1* or *s2*, and then the *n* wide characters from the temporary array are copied into the object pointed to by *s1*.

Example

```
/* This is an example of the function wmemmove(wchar_t*, const wchar_t*,
                                                                  size_t)
   This copies the specified number of characters from string2 to string1.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    wchar_t str1[100], str2[100];
    char *line, *line2;
    wchar_t  *retval;
    size_t count;
    int len;

    line ="What is your name?";
    count =15;
    len = strlen(line);
    retval = wmemmove(str1,str2,count);
    if(retval != NULL) {
     printf("Original line: %s\n", line );
     mbstowcs(str2, line,len);
     printf("Number of characters to move: %d",count);
     printf("\nNew string1: ");
```

```
     fputws(wmemmove(str1, str2, count), stdout);
     fputs("\n", stdout);
    }
    else
        printf("Error\n");
}
```

Output

```
Original line: What is your name?
Number of characters to move: 15
New string1: What is your na
```

See Also

wmemset

Synopsis
#include <**wchar.h**>
wchar_t wmemset(const wchar_t **s***,* **wchar_t** *c*, **size_t** *n*);

Purpose
Copies a single value into specified number of locations in an object.

Return Value
The **wmemset**() function returns the value of *s*.

Parameters
s Integer pointer argument.
c Integer argument.
n Unsigned integer argument.

Description
The **wmemset**() function copies the value of *c* into each of the first *n* wide characters of the object pointed to by *s*.

Example

```
/*  This is an example of wmemset(wchar_t*, wchar_t, size_t).  This function
    copies the specified wide character into the first n positions of str1.
*/
#include <wchar.h>
#include <stdlib.h>

int main() {
    int len;
    size_t count;
    wchar_t *retval, str1[100], c;
    char *line;
    c='r';
    count = 10;

    line = "Hello how are you today?";
    len = strlen(line);
    mbstowcs(str1, line, len);
    retval = wmemset(str1,c, count);
    if(retval != NULL) {
     printf("Original string: %s\n", line);
     printf("Character to replace with: ");
     fputwc(c, stdout);
     printf("\nThe number of characters to replace: %d\n", count);
     printf("The new string is: ");
     fputws(retval, stdout);
```

```
     fputs("\n", stdout);
    }
    else
        printf("Error\n");
}
```

Output

```
Original string: Hello how are you today?
Character to replace with: r
The number of characters to replace: 10
The new string is: rrrrrrrrrrare you today?
```

See Also

第22章　ワイド文字関数 — ＜wctype.h＞

ヘッダー **wctype.h** は、3 つのデータ型、1 つのマクロ、および多くの関数を宣言しています。

宣言されている型は以下のとおりです。

wint_t

は、拡張文字セットのいかなるメンバにも該当しない少なくとも 1 つの値以外に、拡張文字セットのメンバに該当する任意の値も保持可能な既定の引数の上位変換では変更されない整数型（以下の **WEOF** を参照）、

wctrans_t

は、ロケールに特化した文字マッピングを表す値を保持可能なスカラー型、および

wctype_t

は、ロケールに特化した文字分類を表す値を保持可能なスカラー型です。

定義されているマクロは以下のとおりです。

WEOF

は、拡張文字セットのいかなるメンバにも該当しない値をもつ **wint_t** 型の定数表現を拡張します。このマクロは、*end-of-file*、すなわちストリームからの入力がないことををを示すために、この節で説明されているいくつかの関数で受け付けられ（そして返され）ています。このマクロはさらに、拡張文字セットのいかなるメンバにも該当しないワイド文字の値としても使用されます。

宣言されている関数は、以下のようにグループ分けされています。

—ワイド文字分類を提供する関数、

—ワイド文字分類を提供する拡張可能な関数、

—ワイド文字ケースマッピングを提供する関数、

—ワイド文字ケースマッピングを提供する拡張可能な関数。

wint_t 型の引数を受け付ける、この節で説明されているすべての関数に対して、値は **whar_t** として表されるか、またはマクロ **WEOF** の値に等しくすべきです。この引数がこれ以外の値をもつ場合、その結果は不定になります。

これらの関数の実行結果は、現在のロケールの **LC_TYPE** カテゴリに影響されます。

Public データ
なし。

Functions
ヘッダーファイル **wctype.h** では、以下の関数がプロトタイプ化されています。

関数	説明
iswalnum()	数字またはアルファベットのワイド文字であるかどうかをテストします。
iswalpha()	アルファベットのワイド文字であるかどうかをテストします。
iswcntrl()	制御文字のワイド文字であるかどうかをテストします。
iswctype()	文字がワイド文字のプロパティをもつかどうかを定めます。
iswdigit()	10 進数字のワイド文字であるかどうかをテストします。
iswgraph()	出力文字のワイド文字であるかどうかをテストします。
iswlower()	アルファベット小文字のワイド文字であるかどうかをテストします。
iswprint()	出力文字のワイド文字であるかどうかをテストします。
iswpunct()	句読点文字のワイド文字であるかどうかをテストします。
iswspace()	空白文字のワイド文字であるかどうかをテストします。
iswupper()	アルファベット大文字のワイド文字であるかどうかをテストします。
iswxdigit()	16 進文字のワイド文字であるかどうかをテストします。
towctrans()	ワイド文字をマッピングします。
towlower()	アルファベット大文字を小文字に変換します。
towupper()	アルファベット小文字を大文字に変換します。
wctrans()	ワイド文字間のマッピングを記述します。
wctype()	ワイド文字のクラスを記述します。

マクロ
ヘッダー **wctype** では、以下のマクロが定義されています。

マクロ	説明

WEOF *end-of-file* の指定に使用されます。

宣言されている型

wctype ヘッダーでは、以下の型が宣言されています。

宣言されている型	説明
wint_t	int 型 - 拡張セットのメンバに該当する値を保持します。
wctrans_t	スカラー型 - ロケールに特化した文字マッピングを表す値を保持します。
wctype_t	スカラー型 - ロケールに特化した文字分類を表す値を保持します。

移植性

このヘッダーには移植性に関する既知の問題はありません。

iswalnum

Synopsis
#include <wctype.h>
int iswalnum(wint_t *wc***);**

Purpose
Tests to see if a character is a wide alpha-numeric character.

Return Value
The **iswalnum**() function returns a boolean value. It returns true if the character is a wide alpha-numeric character.

Parameters
wc Integer argument.

Description
The **iswalnum**() function tests for any wide character for which **iswalpha**() or **iswdigit**() is true.

Example

```
/* This is an example of the function iswalnum(wint_t ).  This function checks
   if the parameter is an alphanumeric character.  If true, the function
   returns a positive number, or zero if false.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;

    wc = 'i';
    fputwc(wc, stdout);
    if(iswalnum(wc) != 0)
     printf(": is a alphanumeric character.\n");
    else
        printf(": is not an alphanumeric character.\n");
}
```

Output

```
i: is a alphanumeric character.
```

See Also

iswalpha

Synopsis
#include <wctype.h>
int iswalpha(wint_t *wc***);**

Purpose
Tests to see if a character is an upper or lower case alphabetic wide character.

Return Value
The **iswalpha**() function returns a boolean value. It returns true if the character is an upper or lower case alphabetic wide character.

Parameters
wc Integer argument.

Description
The **iswalpha**() function tests for any wide character for which **iswupper**() or **iswlower**() is true, or any wide character that is one of a locale-specific set of alphabetic wide characters for which none of **iswcntrl**(), **iswdigit**(), **iswpunct**(), or **iswspace**() is true.

Example

```
/* This is an example for iswalpha(wint_t).  This function will return a
   non-zero value if the parameter is an alphabetic character.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;

    wc = 'i';
    fputwc(wc, stdout);
    if(iswalpha(wc) > 0)
        printf(": is an alphabetic character\n");
    else
        printf(": is not an alphabetic character\n");
}
```

Output

```
i: is an alphabetic character
```

See Also

iswcntrl

Synopsis
#include <wctype.h>
int iswcntrl(wint_t *wc***);**

Purpose
Tests for a control wide character.

Return Value
The **iswcntrl**() function returns a boolean value. If the character is a control wide character the function returns a true.

Parameters
wc Integer argument.

Description
The **iswcntrl**() function tests for any control wide character.

Example

```
/* This is an example for iswcntrl(wint_t).  This returns a non-zero number
   if it is a control wide character.  If not, it will return zero.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;

    wc = 'i';
    fputwc(wc, stdout);
    if(iswcntrl(wc) > 0)
        printf(": is a wide control character.\n");
    else
        printf(": is not a wide control character.\n");
}
```

Output

```
i: is not a wide control character.
```

See Also

iswctype

Synopsis
#include <**wctype.h**>
int iswctype(wint_t *wc***, wctype_t** *desc***);**

Purpose
Determines whether a character has the properties to be a wide character.

Return Value
The **iswctype**() function returns nonzero (true) if and only if the value of the wide character *wc* has the property described by *desc*.

Parameters
wc Integer argument.
desc Scalar argument.

Description
The **iswctype**() function determines whether the wide character *wc* has the property described by *desc*. The current setting of the **LC_CTYPE** category shall be the same as during the call to **wctype**() that returned the value *desc*.

Each of the following expressions has a truth-value equivalent to the call to the wide-cha racter classification function in the comment that follows the expression:

iswctype(*wc***, wctype(“alnum”)) // iswgraph(***wc***)**
iswctype(*wc***, wctype(“graph”)) // iswgraph(***wc***)**
iswctype(*wc***, wctype(“cntrl”)) // iswcntrl(***wc***)**
iswctype(*wc***, wctype(“digit”)) // iswdigit(***wc***)**
iswctype(*wc***, wctype(“graph”)) // iswgraph(***wc***)**
iswctype(*wc***, wctype(“lower”)) // iswlower(***wc***)**
iswctype(*wc***, wctype(“print”)) // iswprint(***wc***)**
iswctype(*wc***, wctype(“punct”)) // iswpunct(***wc***)**
iswctype(*wc***, wctype(“space”)) // iswspace(***wc***)**
iswctype(*wc***, wctype(“upper”)) // iswupper(***wc***)**
iswctype(*wc***, wctype(“xdigit”)) // iswxdigit(***wc***)**

Example

```
/* This is an example for iswctype(wint_t, wctype_t).  This test the wint_t
   parameter for the specifed property.  If the wide character has the
   specified property, than a nonzero value will be returned.  If it does not,
```

```
   then zero will be returned.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;
    char *prop;

    prop = "upper";
    wc = 'I';
    fputs("The wide character '", stdout);
    fputwc(wc, stdout);
    if(iswctype(wc, wctype(prop)) > 0)
        printf("' has the property: %s\n", prop);
    else
        printf("' does not have the property: %s\n", prop);
}
```

Output

```
The wide character 'I' has the property: upper
```

See Also

iswdigit

Synopsis
#include <**wctype.h**>
int iswdigit(wint_t *wc***);**

Purpose
Tests for decimal-digit wide characters.

Return Value
The **iswdigit**() function returns a boolean value which is true when the character is a decimal-digit character.

Parameters
wc Integer argument.

Description
The **iswdigit**() function tests for any wide character that corresponds to a decimal-digit character.

Example

```
/* This is an example for iswdigit(wint_t).  This function will return a
   non-zero value if the parameter is digit.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;

    wc = '1';
    fputwc(wc, stdout);
    if(iswdigit(wc) > 0)
        printf(": is a digit.\n");
    else
        printf(": is not a digit.\n");
}
```

Output

```
1: is a digit.
```

See Also

iswgraph

Synopsis
#include <**wctype.h**>
int iswgraph(wint_t *wc***);**

Purpose
Tests for printing wide characters.

Return Value
The **iswgraph**() function returns a boolean value which is true when the character is a printing wide character.

Parameters
wc Integer argument.

Description
The **iswgraph**() function tests for any wide character for which **iswprint**() is true and **iswspace**() is false.

Example

```
/* This is an example for iswgraph(wint_t).  This function will return a
   non-zero value if the parameter is a printable character except space.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;

    wc = 'i';
    fputwc(wc, stdout);
    if(iswgraph(wc) != 0)
printf(": is a printable character.\n");
    else
 printf(": is not a printable character.\n");
}
```

Output

```
i: is a printable character.
```

See Also

iswlower

Synopsis
#include <wctype.h>
int iswlower(wint_t *wc***);**

Purpose
Tests for lowercase wide characters.

Return Value
The **iswlower**() function returns a boolean value which is true when the argument is a lower case letter.

Parameters
wc Integer argument.

Description
The **iswlower**() function tests for any wide character that correspnds to a lowercase letter or is one of a locale-specific set of wide characters for which none of **iswcntrl**(), **iswdigit**(), **iswpunct**(), or **iswspace**() is true.

Example

```
/* This is an example for iswlower(wint_t).  This function will return a
   non-zero value if the parameter is a lowercase letter.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;

    wc = 'i';
    fputwc(wc, stdout);
    if(iswlower(wc) != 0)
         printf(": is a lowercase letter.\n");
    else
          printf(": is not a lowercase letter.\n");
}
```

Output

```
i: is a lowercase letter.
```

See Also

iswprint

Synopsis
#include <**wctype.h**>
int iswprint(wint_t *wc***);**

Purpose
Tests for printing wide characters.

Return Value
The **iswprint**() function returns a boolean value which is true when the argument is a printing wide character.

Parameters
wc Integer argument.

Description
The **iswprint**() function tests for any printing wide character.

Example

```
/* This is an example for iswprint(wint_t).  This function will return a
   non-zero value if the parameter is a printable character including space
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;

    wc = 'i';
    fputwc(wc, stdout);
    if(iswprint(wc) != 0)
        printf(": is a printable character.\n");
    else
        printf(": is not a printable character.\n");
}
```

Output

```
i: is a printable character.
```

See Also

iswpunct

Synopsis
#include <wctype.h>
int iswpunct(wint_t *wc***);**

Purpose
Tests for punctuation wide characters.

Return Value
The **iswpunct**() function returns a boolean value which is true when the argument is a punctuation wide character.

Parameters
wc Integer argument.

Description
The **iswpunct**() function tests for any printing wide character that is one of a locale-specific set of punctuation wide characters for which neither **iswspace**() nor **iswalnum**() is true.

Example

```
/* This is an example for iswpunct(wint_t).  This function will return a
   non-zero value if the parameter is x punctuation character.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;

    wc = '.';
    fputwc(wc, stdout);
    if(iswpunct(wc) != 0)
        printf(": is a punctuation mark.\n");
    else
        printf(": is not a punctuation mark.\n");
}
```

Output

```
.: is a punctuation mark.
```

See Also

iswspace

Synopsis
#include <wctype.h>
int iswspace(wint_t *wc***);**

Purpose
Tests for white-space wide characters.

Return Value
The **iswspace**() function returns a boolean value which is true when the argument is a white-space wide characters.

Parameters
wc Integer argument.

Description
The **iswspace**() function tests for any wide character that corresponds to a locale-specific set of white-space wide characters for which none of **iswalnum**(), **iswgraph**(), or **iswpunct**() is true.

Example

```
/* This is an example for iswspace(wint_t).  This function will return a
   non-zero value if the parameter is a white space character.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;

    wc = ' ';
    fputwc(wc, stdout);
    if(iswspace(wc) != 0)
        printf(": is a white space character.\n");
    else
        printf(": is not a white space character.\n");
}
```

Output

```
 : is a white space character.
```

See Also

iswupper

Synopsis
#include <wctype.h>
int iswupper(wint_t *wc***);**

Purpose
Tests for uppercase letter wide characters.

Return Value
The **iswupper**() function returns a boolean value which is true when the argument is an uppercase letter wide character.

Parameters
wc Integer argument.

Description
The **iswupper**() function tests for any wide character that corresponds to an uppercase letter or is one of a locale-specific set of wide characters for which none of **iswcntrl**(), **iswdigit**(), **iswpunct**(), or **iswspace**() is true.

Example

```
/* This is an example for the function iswupper(wint_t).  This function returns
   a non negative number if the input is an uppercase letter.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;

    wc = 'I';
    fputwc(wc, stdout);
    if(iswupper(wc) != 0)
        printf(": is an uppercase letter.\n");
    else
        printf(": is not an uppercase letter.\n");
}
```

Output

```
I: is an uppercase letter.
```

See Also

iswxdigit

Synopsis
#include <wctype.h>
int iswxdigit(wint_t *wc***);**

Purpose
Tests for hexadecimal-digit wide characters.

Return Value
The **iswxdigit**() function returns a boolean value which is true when the argument is a hexadecimal-digit character.

Parameters
wc Integer argument.

Description
The **iswxdigit**() function tests for any wide character that corresponds to a hexadecimal-digit character.

Example

```
/* This is an example for iswxdigit(wint_t).  This function will return a
   non-zero value if the parameter is a hexadecimal digit.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;

    wc = 'A';
    fputwc(wc, stdout);
    if(iswxdigit(wc) != 0)
        printf(": is a hexadecimal digit.\n");
    else
        printf(": is not a hexadecimal digit.\n");
}
```

Output

```
A: is a hexadecimal digit.
```

See Also

towctrans

Synopsis
#include <towctrans.h>
wint_t towctrans(wint_t *wc***);**

Purpose
Maps wide characters.

Return Value
The **towctrans**() function returns the mapped value of *wc* using the mapping described by *desc*.

Parameters
wc Integer argument.
desc Scalar argument.

Description
The **towctrans**() function maps the wide character *wc* using the mapping described by *desc*. The current setting of the **LC_TYPE** category shall be the same as during the call to **wctrans**() that returned the value *desc*.

Each of the following expressions behaves the same as the call to the wide-character case- mapping function in the comment that follows the expression:

towctrans(*wc***, wctrans("tolower")) /* towlower(***wc***) */**
towctrans(*wc***, wctrans("toupper")) /* towupper(***wc***) */**

Example

```
/* This is an example for towctrans(wint_t, wctrans_t).  This test the wint_t
   parameter for the specifed property.  If the wide character has the
   specified property, than a nonzero value will be returned.  If it does not,
   then zero will be returned.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;
    char *prop;

    prop = "toupper";
    wc = 'i';
    fputs("The wide character '", stdout);
    fputwc(wc, stdout);
    printf("' has been changed to: '");
```

```
    fputwc(towctrans(wc, wctrans(prop)), stdout);
    printf("'. \nBy the property of %s.\n", prop);
}
```

Output

```
The wide character 'i' has been changed to: 'I'.
By the property of toupper.
```

See Also

towlower

Synopsis
#include <**towlower.h**>
wint_t towlower(wint_t *wc***);**

Purpose
Converts uppercase letter to corresponding lowercase letter.

Return Value
If the argument is a wide character for which **iswupper**() is true and there are one or more corresponding wide characters, as specified by the current locale , for which **iswlower**() is true, the **towlower**() funciton returns one of the correspoding wide characters; otherwise, the argument is returned unchanged.

Parameters
wc Integer argument.

Description
The **towlower**() function converts an uppercase letter to a corresponding lowercase letter.

Example

```
/* This is an example of the function towlower(wint_t).  This will convert the
   wide character into its lowercase representation.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;
    wc ='A';

    printf("Before: ");
    fputwc(wc, stdout);
    printf("\n");
    printf("After: ");
    fputwc(towlower(wc), stdout);
}
```

Output

```
Before: A
After: a
```

See Also

towupper

Synopsis
#include <**towupper.h**>
wint_t towupper(wint_t *wc***);**

Purpose
Converts lowercase letter to corresponding uppercase letter.

Return Value
If the argument is a wide character for which **iswlower**() is true and there are one or more corresponding wide characters, as specified by the current locale , for which **iswupper**() is true, the **towupper**() function returns one of the correspoding characters; otherwise, the argument is returned unchanged.

Parameters
wc Integer argument.

Description
The **towupper**() function converts a lowercase letter to a corresponding uppercase letter.

Example

```
/* This is an example of the function towupper(wint_t).  This function converts
   lowercase letters to uppercase.
*/
#include <wctype.h>
#include <wchar.h>

int main() {
    wint_t wc;
    wc = 'i';

    printf("Before: ");
    fputwc( wc, stdout);
    printf("\n");
    printf("After: ");
    fputwc( towupper(wc), stdout);
    printf("\n");
}
```

Output

```
Before: i
After: I
```

See Also

wctrans

Synopsis
#include <wctype.h>
wctrans_t wctrans(const char ***property*****);**

Purpose
Describes mapping between wide characters.

Return Value
If *property* identifies a valid mapping of wide characters according to the **LC_CTYPE** category of the current locale, the **wctrans**() function returns a nonzero value that is valid as the second argument to the **towctrans**() function; otherwise, it returns zero.

Parameters
**property* Character argument.

Description
The **wctype**() function constructs a value with type **wctrans_t**() that describes a mapping between wide characters identified by the string argument **property**.

The strings listed in the description of the **towctrans**() function shall be valid in all locales as *property* arguments to the **wctrans**() function.

Example

```
/* This is an example of the function wctrans(const char*).  This returns a
   non-negative number if the string is a valid wctrans.
*/
#include <wctype.h>

int main() {
    char *prop;
    prop ="tolower";
    if(wctrans(prop) > 0)
printf("'%s' is a valid wctrans.\n", prop);
    else
        printf("'%s' is not a valid wctrans.\n", prop);
}
```

Output

```
'tolower' is a valid wctrans.
```

See Also

wctype

Synopsis
#include <**wctype.h**>
wctype_t wctype(const char **property**);*

Purpose
Describes a class of wide characters.

Returns
If *property* identifies a valid class of wide characters according to the **LC_CTYPE** category of the current locale, the **wctype**() function returns a nonzero value that is valid as the second argument to the **iswctype**() function; otherwise, it returns zero.

Parameters
**property* Character argument.

Description
The **wctype**() function constructs a value with type **wctype_t**() that describes a class of wide characters identified by the string argument *property*.

The strings listed in the description of the **iswctype**() function shall be valid in all locales as *property* arguments to the **wctype**() function.

Example

```
/* This is an example of the function wctype(const char*).  This returns a
   non-negative number if the string is a valid wctype.
*/
#include <wctype.h>

int main() {
    char *prop;
    prop ="xdigit";
    if(wctype(prop) > 0)
        printf("'%s' is a valid wctype.\n", prop);
    else
        printf("'%s' is not a valid wctype.\n", prop);
}
```

Output

```
'xdigit' is a valid wctype.
```

See Also
iswctype()

付 録A　特定のプラットフォームでサポートされていない関数

Ch のほとんどの関数は、異なる複数のプラットフォームでサポートされています。特定のオペレーティングシステムでサポートされていない関数を以下に示します。

A.1　HPUX

HPUX オペレーティングシステムでリターンエラーコードのみが実装されている関数を以下に示します

関数	ヘッダーファイル: windows/winsock.h	戻り値
HANDLE WSAAPI WSAAsyncGetHostByAddr(**HWND** *hwnd*, **unsigned int** *wMsg*, **const char FAR** **addr*, **int** *len*, **int** *type*, **char FAR** **buf*, **int** *buflen*)		0
HANDLE WSAAPI WSAAsyncGetHostByName(**HWND** *hwnd*, **unsigned int** *wMsg*, **const char FAR** **name*, **char FAR** **buf*, **int** *buflen*)		0
HANDLE WSAAPI WSAAsyncGetProtoByName(**HWND** *hwnd*, **unsigned int** *wMsg*, **const char FAR** **name*, **char FAR** **buf*, **int** *buflen*)		0
HANDLE WSAAPI WSAAsyncGetProtoByNumber(**HWND** *hwnd*, **unsigned int** *wMsg*, **int** *number*, **char FAR** **buf*, **int** *buflen*)		0
HANDLE WSAAPI WSAAsyncGetServByName(**HWND** *hwnd*, **unsigned int** *wMsg*, **const char FAR** **name*, **const char FAR** **proto*, **char FAR** **buf*, **int** *buflen*)		0
HANDLE WSAAPI WSAAsyncGetServByPort(**HWND** *hwnd*, **unsigned int** *wMsg*, **int** *port*, **const char FAR** **proto*, **char FAR** **buf*, **int** *buflen*)		0
int WSAAPI WSAAsyncSelect(**SOCKET** *s*, **HWND** *hwnd*, **unsigned int** *wMsg*, **long** *lEvent*)		– 1
int WSAAPI WSACancelAsyncRequest(**HANDLE** *AsyncTaskHandle*)		– 1
int WSAAPI WSACancelBlockingCall(**void**)		– 1
int WSAAPI WSACleanup(**void**)		0
void WSAAPI WSAGetLastError(**void**)		N/A
BOOL WSAAPI WSAIsblocking(**void**)		False

BOOL WSAAPI WSASetBlockingHook(**FARPROC** *lpBlockFunc*)	NULL
void WSAAPI WSASetLastError(**int** *iError*)	N/A
int WSAAPI WSAStartup(**WORD** *wVersionRequested*, **LPWSADATA** *lpWSAData*)	0
int WSAAPI WSAUnhookBlockingHook(**void**)	– 1
int WSAAPI ioctlsocket(**SOCKET** *s*, **long** *cmd*, **u_long** **argp*)	– 1

関数	ヘッダーファイル: netdb.h	戻り値
struct hostent *gethostbyname_r(**const char** **name*, **struct hostent** **result*, **char** **buffer*, **int** *buflen*, **int** **h_errnop*)		NULL
struct hostent *gethostbyaddr_r(**const char** **addr*, **int** *length*, **int** *type*, **struct hosteni** **result*, **char** **buffer*, **int** *buflen*, **int** **h_errnop*)		NULL
struct hostent *gethostent_r(**struct hostent** **result*, **char** **buffer*, **int** *buflen*, **int** **h_errnop*)		NULL
struct servent *getservbyname_r(**const char** **name*, **const char** **proto*, **struct servent** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct servent *getservbyport_r(**int** *port*, **const char** **proto*, **struct servent** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct servent *getservent_r(**struct hosten** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct netent *getnetbyname_r(**const char** **name*, **struct netent** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct netent *getnetbyaddr_r(**long** *net*, **int** *type*, **struct netent** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct netent *getnetent_r(**struct netent** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct protoent *getprotobyname_r(**const char** **name*, **struct protent** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct protoent *getprotobynumber_r(**int** *proto*, **struct protent** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct protoent *getprotoent_r(**struct protent** **result*, **char** **buffer*, **int** *buflen*)		NULL

関数	ヘッダーファイル: math.h	戻り値
double gamma_r(**double** *x*, **int** **signagamp*)		– 1
double scalbn(**double** *x*, **int** **n*)		– 1
int isnormal(*real-floating x*)		0
int signbit(*real-floating x*)		– 1

関数	ヘッダーファイル: unistd.h	戻り値
char getlogin_r(char **name*, **int** *namelen*)		NULL
char *ttyname_r(int *fildes*, **char** **buf*, **int** *len*)		NULL
int setegid(uid_t *egid*)		0
int seteuid(uid_t *egid*)		0

関数	ヘッダーファイル: grp.h	戻り値
struct group *getgrnam_r(const char **name*, **struct group** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct group *getgrgid_r(gid_t *gid*, **struct group** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct group *fgetgrent_r(FILE **f*, **struct group** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct group *fgetgrent(FILE **f*)		NULL

関数	ヘッダーファイル: mqueue.h	戻り値
ssize_t mq_receive(mqd_t *mqdes*, **char** **msg_ptr*, **size_t** *msq_len*, **unsigned int** **msg_prio*)		0

関数	ヘッダーファイル: sys/procset.h	戻り値
int sigsendset(procset_t **psp*, **int** *sig*)		– 1

関数	ヘッダーファイル: libintl.h	戻り値
char *gettext(const char **msgid*)		NULL
char *dgettext(const char **domaninname*, **const char** **msgid*)		NULL
char *textdomain(const char **domainname*)		NULL
char *bindtextdomain(const char **domainname*)		NULL

関数	ヘッダーファイル: time.h	戻り値
int ascftime(**char** **s*, **const char** **format*, **const struct tm** **timeptr*)		0
int cftime(**char** **s*, **const char** **format*, **const time_t** *clock*)		0

関数	ヘッダーファイル: wchar.h	戻り値
wint_t btowc(**int** *c*)		WEOF
int fwide(**FILE** **stream*, **int** *c*)		0
int mbsinit(**const mbstate_t** **ps*)		0
size_t mbrlen(**const char** **str*, **size_t** *length*, **mbstate_t** *ps*)		0
size_t mbrtowc(**wchar_t** **str1*, **const char** **str2*, **size_t** *length*, **mbstate_t** *ps*)		– 1
size_t mbsrtowcs(**wchar_t** **str1*, **const char** ***str2*, **size_t** *length*, **mbstate_t** **ps*)		– 1
size_t wcrtomb(**char** **str1*, **wchar_t** *s*, **mbstate_t** **ap*)		– 1
size_t wcsrtombs(**char** **str1*, **const char** ***str2*, **size_t** *length*, **mbstate_t** **ps*)		– 1
wchar_t wcsstr(**const wchar_t** **str1*, **const wchar_t** **str2*)		NULL
int wctob(**wint_t** *wc*)		EOP
wchar_t wmemchr(**const wchar_t** **str1*, **wchar_t** *c*, **size_t** *count*)		NULL
int wmemcmp(**const wchar_t** **str1*, **const wchar_t** **str2*, **size_t** *count*)		NULL
wchar_t *wmemmove(**wchar_t** **str1*, **const wchar_t** **str2*, **size_t** *count*)		NULL
wchar_t *wmemset(**wchar_t** **str1*, **wchar_t** *str2*, **size_t** *count*)		NULL
wchar_t *wmemcpy(**wchar_t** **str1*, **const wchar_t** **str2*, **size_t** *count*)		NULL

関数	ヘッダーファイル: wctype.h	戻り値
wint_t towctrans(**win_t** *wc*, **wctrans_t** *desc*)		– 1
wctrans_t wctrans(**const char** **prop*)		0

A.2 Linux

Linux オペレーティングシステムでリターンエラーコードのみが実装されている関数を以下に示します。

関数	ヘッダーファイル: windows/winsock.h	戻り値

HANDLE WSAAPI WSAAsyncGetHostByAddr(**HWND** *hwnd*, **unsigned int** *wMsg*, **const char FAR** **addr*, **int** *len*, **int** *type*, **char FAR** **buf*, **int** *buflen*)	0
HANDLE WSAAPI WSAAsyncGetHostByName(**HWND** *hwnd*, **unsigned int** *wMsg*, **const char FAR** **name*, **char FAR** **buf*, **int** *buflen*)	0
HANDLE WSAAPI WSAAsyncGetProtoByName(**HWND** *hwnd*, **unsigned int** *wMsg*, **const char FAR** **name*, **char FAR** **buf*, **int** *buflen*)	0
HANDLE WSAAPI WSAAsyncGetProtoByNumber(**HWND** *hwnd*, **unsigned int** *wMsg*, **int** *number*, **char FAR** **buf*, **int** *buflen*)	0
HANDLE WSAAPI WSAAsyncGetServByName(**HWND** *hwnd*, **unsigned int** *wMsg*, **const char FAR** **name*, **const char FAR** **proto*, **char FAR** **buf*, **int** *buflen*)	0
HANDLE WSAAPI WSAAsyncGetServByPort(**HWND** *hwnd*, **unsigned int** *wMsg*, **int** *port*, **const char FAR** **proto*, **char FAR** **buf*, **int** *buflen*)	0
int WSAAPI WSAAsyncSelect(**SOCKET** *s*, **HWND** *hwnd*, **unsigned int** *wMsg*, **long** *lEvent*)	– 1
int WSAAPI WSACancelAsyncRequest(**HANDLE** *AsyncTaskHandle*)	– 1
int WSAAPI WSACancelBlockingCall(**void**)	– 1
int WSAAPI WSACleanup(**void**)	0
void WSAAPI WSAGetLastError(**void**)	N/A
BOOL WSAAPI WSAIsblocking(**void**)	False
BOOL WSAAPI WSASetBlockingHook(**FARPROC** *lpBlockFunc*)	NULL
void WSAAPI WSASetLastError(**int** *iError*)	N/A
int WSAAPI WSAStartup(**WORD** *wVersionRequested*, **LPWSADATA** *lpWSAData*)	0
int WSAAPI WSAUnhookBlockingHook(**void**)	– 1
int WSAAPI ioctlsocket(**SOCKET** *s*, **long** *cmd*, **u_long** **argp*)	– 1

関数	ヘッダーファイル: aio.h	戻り値
int aio_read(**struct aiocb** **aiocbp*)		– 1
int aio_write(**struct aiocb** **aiocbp*)		– 1
int lio_listio(**int** *mode*, **struct aiocb** **list []*, **int** *nent*, **struct sigevent** **sig*)		– 1
int aio_error(**const struct aiocb** **aiocbp*)		– 1
int aio_return(**struct aiocb** **aiocbp*)		– 1
int aio_cancel(**int** *fildes*, **struct aiocb** **aiocbp*)		– 1
int aio_suspend(**const struct aiocb** **aiocbp[]*, **int** *nent*, **const struct timespec** **timeout*)		– 1

int aio_fsync(int *op,* **struct aiocb** **aiocbp*)	– 1

関数	ヘッダーファイル: time.h	戻り値
char **ctime_r(const time_t** **clock*, **char** **buf*, **int** *buflen*)		NULL
char **cftime(char** **s*, **char** **format*, **const time_t** **clock*)		0
struct tm **localtime_r(const time_t** **clock*, **struct tm** **res*)		NULL
struct tm **gmtime_r(const time_t** **clock*, **struct tm** **res*)		NULL
char **asctime_r(const struct** *tm* **tm*, **char** **buf*, **int** *buflen*)		NULL
int clock_settime(clockid_t *clock_id*, **const struct timespec** **tp*)		– 1
int clock_gettime(clockid_t *clock_id*, **const struct timespec** **tp*)		– 1
int clock_getres(clockid_t *clock_id*, **const struct timespec** **res*)		– 1
int timer_create(clockid_t *clock_id*, **struct sigevent** **evp*, **timer_t** **timerid*)		– 1
int timer_delete(timer_t *timerid*)		– 1
int timer_settime(timer_t *timerid*, **int** *flags*, **const struct itimerspec** **value*, **struct itimerspec** **ovalue*)		– 1
int timer_gettime(timer_t *timerid*, **struct timespec** **value*)		– 1
int timer_getoverrun(timer_t *timerid*)		– 1
int nanosleep(const struct timespec **rqtp*, **struct timespec** **rmtp*)		– 1
char ascftime(char **s*, **const char** **format*, **const struct tm** **timeptr*)		0

関数	ヘッダーファイル: libgen.h	戻り値
char* **regcmp(const char** **string1*, /* **char***string2* */, ... /* (**char***)0 */)		NULL
char* **regex(const char** **re*, **const char** **subject*, /* **char***ret0* */ ...)		NULL

関数	ヘッダーファイル: math.h	戻り値
int fpclassify(*real-floating x*)		– 1
int isnormal(*real-floating x*)		0
double gamma_r(double *x*, **int** **signgamp*)		NaN
int signbit(*real-floating x*)		– 1

関数	ヘッダーファイル: signal.h	戻り値
int sighold(**int** *sig*)		– 1
int sigrelse(**int** *sig*)		– 1
int sigignore(**int** *sig*)		– 1
int sigaltstack(**const stack_t** *ss*, **stack_t** *oss*)		– 1

関数	ヘッダーファイル: stropts.h	戻り値
int fattach(**int** *fildes*, **const char** *path*)		– 1
int fdetach(**const char** *path*)		– 1
int getmsg(**int** *fildes*, **struct strbuf** *ctptr*, **struct strbuf** *dataptr*, **long** *flagsp*)		– 1
int getpmsg(**int** *fildes*, **struct strbuf** *ctptr*, **struct strbuf** *dataptr*, **int** *band*, **long** *flagsp*)		– 1
int isastream(**int** *fildes*)		– 1
int putmsg(**int** *fildes*, **struct strbuf** *ctptr*, **struct strbuf** *dataptr*, **long** *flags*)		– 1
int putpmsg(**int** *fildes*, **struct strbuf** *ctptr*, **struct strbuf** *dataptr*, **int** *band*, **long** *flags*)		– 1

関数	ヘッダーファイル: sys/mman.h	戻り値
int msync(**void** * *addr*, **size_t** *len*, **int** *flags*)		– 1
int mlock(**void** *addr, **size_t** *len*)		– 1
int munlock(**void** *addr, **size_t** *len*)		– 1
int shm_open(**const char** *name, **int** *oflag*, **mode_t** *mode*)		– 1
int shm_unlink(**const char** *name*)		– 1

関数	ヘッダーファイル: sys/procset.h	戻り値
int sigsend(**idtype_t** *idtype*, **id_t** *id*, **int** *sig*)		– 1
int sigsendset(**procset_t** *psp*, **int** *sig*)		– 1

関数	ヘッダーファイル: sys/wait.h	戻り値

int waitid(**idtype_t** *idtype*, **id_t** *id*, **siginfo_t** **infop*, **int** *options*)	– 1

関数	ヘッダーファイル: poll.h	戻り値
int poll(**struct pollfd** **fds*, **unsigned int** *nfds*, **int** *timeout*)		– 1

関数	ヘッダーファイル: unistd.h	戻り値
int fchroot(**int** *fildes*)		– 1
char ***gettxt(** **const char** **msgid*, **const char** **dflt_str*)		NULL
pid_t getsid(**pid_t** *pid*)		– 1
int mincore(**caddr_t** *addr*, **size_t** *len*, **char** **vec*) **unsigned int** *scale*)		0

A.3 Solaris

Solaris オペレーティングシステムでリターンエラーコードのみが実装されている関数を以下に示します。

関数	ヘッダーファイル: windows/winsock.h	戻り値
HANDLE WSAAPI WSAAsyncGetHostByAddr(**HWND** *hwnd*, **unsigned int** *wMsg*, **const char FAR** **addr*, **int** *len*, **int** *type*, **char FAR** **buf*, **int** *buflen*)		0
HANDLE WSAAPI WSAAsyncGetHostByName(**HWND** *hwnd*, **unsigned int** *wMsg*, **const char FAR** **name*, **char FAR** **buf*, **int** *buflen*)		0
HANDLE WSAAPI WSAAsyncGetProtoByName(**HWND** *hwnd*, **unsigned int** *wMsg*, **const char FAR** **name*, **char FAR** **buf*, **int** *buflen*)		0
HANDLE WSAAPI WSAAsyncGetProtoByNumber(**HWND** *hwnd*, **unsigned int** *wMsg*, **int** *number*, **char FAR** **buf*, **int** *buflen*)		0
HANDLE WSAAPI WSAAsyncGetServByName(**HWND** *hwnd*, **unsigned int** *wMsg*, **char FAR** **buf*, **int** *buflen*)		0
HANDLE WSAAPI WSAAsyncGetServByPort(**HWND** *hwnd*, **unsigned int** *wMsg*, **int** *port*, **const char FAR** **proto*, **char FAR** **buf*, **int** *buflen*)		0
int WSAAPI WSAAsyncSelect(**SOCKET** *s*, **HWND** *hwnd*, **unsigned int** *wMsg*,		– 1

long *lEvent*)	
int WSAAPI WSACancelAsyncRequest(HANDLE *AsyncTaskHandle*)	– 1
int WSAAPI WSACancelBlockingCall(void)	– 1
int WSAAPI WSACleanup(void)	0
void WSAAPI WSAGetLastError(void)	N/A
BOOL WSAAPI WSAIsblocking(void)	False
BOOL WSAAPI WSASetBlockingHook(FARPROC *lpBlockFunc*)	NULL
void WSAPPI WSASetLastError(int *iError*)	N/A
int WSAAPI WSAStartup(WORD *wVersionRequested*, **LPWSADATA** *lpWSAData*)	0
int WSAAPI WSAUnhookBlockingHook(void)	– 1
int WSAAPI ioctlsocket(SOCKET *s*, **long** *cmd*, **u_long** **argp*)	– 1

関数	ヘッダーファイル: math.h	戻り値
int fpclassify(*real-floating x*)		– 1
int isnormal(double *x*)		0
int signbit(double *x*)		0

関数	ヘッダーファイル: wchar.h	戻り値
wint_t btowc(int *c*)		WEOF
int fwide(FILE **stream*, **int** *c*)		0
int mbsinit(const mbstate_t **ps*)		0
size_t mbrlen(const char **str*, **size_t** *length*. **mbstate_t** *ps*)		0
size_t mbrtowc(wchar_t **str1*, **const char** **str2*, **size_t** *length*, **mbstate_t** *ps*)		– 1
size_t mbsrtowcs(wchar_t **str1*, **const char** ***str2*, **size_t** *length*, **mbstate_t** **ps*)		– 1
size_t wcrtomb(char **str1*, **wchar_t** *s*, **mbstate_t** **ap*)		– 1
size_t wcsrtombs(char **str1*, **const char** ***str2*, **size_t** *length*, **mbstate_t** **ps*)		– 1
wchar_t **wcsstr(const wchar_t* **str1*, **const wchar_t** **str2*)		NULL
int wctob(wint_t *wc*)		EOP
wchar_t wmemchr(const wchar_t **str1*, **wchar_t** *c*, **size_t** *count*)		NULL
int wmemcmp(const wchar_t **str1*, **const wchar_t** **str2*, **size_t** *count*)		NULL
wchar_t wmemmove(wchar_t **str1*, **const wchar_t** **str2*, **size_t** *count*)		NULL
wchar_t wmemset(wchar_t **str1*, **wchar_t** *str2*, **size_t** *count*)		NULL
wchar_t wmemcpy(wchar_t **str1*, **const wchar_t** **str2*, **size_t** *count*)		NULL

A.4 Windows

Windows 95/98/NT オペレーティングシステムでリターンエラーコードのみが実装されている関数を以下に示します。

関数	ヘッダーファイル: arpa/inet.h	戻り値
unsigned long inet_network(**const char** *cp*)		– 1
struct in_addr inet_makeaddr(**int** *net*, **int** *lna*)		– 1
unsigned long inet_lnaof(**struct in_addr** *in*)		– 1
unsigned long inet_netof(**struct in_addr** *in*)		– 1

関数	ヘッダーファイル: libintl.h	戻り値
char ***gettext(** **const char** **msgid*)		NULL
char ***dgettext(** **const char** **domaninname*, **const char** **msgid*)		NULL
char ***textdomain(** **const char** **domainname*)		NULL
char ***bindtextdomain(** **const char** **domainname*)		NULL

関数	ヘッダーファイル: aio.h	戻り値
int aio_read(**struct aiocb** **aiocbp*)		– 1
int aio_write(**struct aiocb** **aiocbp*)		– 1
int lio_listio(**int** *mode*, **struct aiocb** **list []*, **int** *nent*, **struct sigevent** **sig*)		– 1
int aio_error(**const struct aiocb** **aiocbp*)		– 1
int aio_return(**struct aiocb** **aiocbp*)		– 1
int aio_cancel(**int** *fildes*, **struct aiocb** **aiocbp*)		– 1
int aio_suspend(**const struct aiocb** **aiocbp[]*, **int** *nent*, **const struct timespec** **timeout*)		– 1
int aio_fsync(**int** *op*, **struct aiocb** **aiocbp*)		– 1

関数	ヘッダーファイル: dirent.h	戻り値
int closedir(**DIR** **dirp*)		– 1
DIR opendir(**const char** **filename*)		NULL

struct dirent readdir(**DIR** **dirp*)	NULL
int readdir_r(**DIR** **dirp*, **struct dirent** **entry*, **struct dirent** ***result*)	NULL
void rewinddir(**DIR** **dirp*)	N/A
void seekdir(**DIR** **dirp*, **long** *loc*)	N/A
long telldir(**DIR** **dirp*)	– 1

関数	ヘッダーファイル: glob.h	戻り値
int glob(**const char** **pattern*, **int** *flags*, **int** (**errfunc*(**const char** **epath*, **int** *eerrno*) **glob_t** **pglob*)		–1
void globfree(**glob_t** **pglob*)		N/A

関数	ヘッダーファイル: grp.h	戻り値
int initgroups(**const char** **name*, **gid_t** *basegid*)		– 1
void setgrent(**void**)		N/A
struct group ***getgrent**(**void**)		NULL
int getgrent_r(**struct group** **result*, **char** **buffer*, **int** *buflen*, **FILE** ***grfp*)		– 1

関数	ヘッダーファイル: libgen.h	戻り値
char* **regcmp**(**const char** **string1*, /* **char***string2* */, ... /* (**char***)0 */)		NULL
char* **regex**(**const char** **re*, **const char** **subject*, /* **char***ret0* */ ...)		NULL

関数	ヘッダーファイル: math.h	戻り値
double erf(**double** *x*)		NaN
double erfc(**double** *x*)		NaN
double cbrt(**double** *x*)		NaN
double rint(**double** *x*)		NaN
double scalbn(**double** *x*, **int** *n*)		NaN
double remainder(**float** *x*, **float** *x*)		NaN
double ilogb(**double** *x*)		NaN
double gamma(**double** *x*)		NaN
double lgamma(**double** *x*)		NaN

double gamma_r(**double** *x*, **int** **signgamp*)	NaN
double lgammar(**double** *x*, **int** **signgamp*)	NaN
int fpclassify(*real-floating x*)	– 1
int isnormal(*real-floating x*)	0
int signbit(*real-floating x*)	– 1

関数	ヘッダーファイル: netdb.h	戻り値
struct hostent *gethostent(**void**)		NULL
struct netent *getnetbyname(**const char** **name*)		NULL
struct netent *getnetbyaddr(**long** *net*, **int** *type*)		NULL
struct netent *getnetent(**void**)		NULL
struct servent *getservent(**void**)		NULL
struct protoent *getprotoent(**void**)		NULL
int sethostent(**int** *stayopen*)		– 1
int endhostent(**void**)		– 1
int endnetent (**void**)		– 1
int setnetent(**int** *stayopen*)		– 1
int setservent(**int** *stayopen*)		– 1
int endservent(**void**)		– 1
int setprotoent(**int** *stayopen*)		– 1
int endprotoent(**void**)		– 1
int rcmd(**char** ***ahost*, **unsigned short** *inport*, **char** **luser*, **char** **ruser*, **char** **cmd*, **int** **fd2p*)		– 1
int rresvport(**int** **port*)		– 1
int ruserok(**char** **rhost*, **int** *suser*, **char** **ruser*, **char** **luser*)		– 1
int rexec(**char** ***ahost*, **unsigned short** *inport*, **char** **user*, **char** **passwd*, **char** **cmd*, **int** **fd2p*)		– 1
struct hostent *gethostbyname_r(**const char** **name*, **struct hostent** **result*, **char** **buffer*, **int** **buflen*, **int** **h_errnop*)		NULL
struct hostent *gethostbyaddr_r(**const char** **addr*, **int** *length*, **int** *type*, **struct hostent** **result*, **char** **buffer*, **int** *buflen*, **int** **h_errnop*)		NULL
struct hostent *gethostent_r(**struct hostent** **result*, **char** **buffer*, **int** *buflen*, **int** **h_errnop*)		NULL
struct servent *getservbyname_r(**const char** **name*, **const char** **proto*, **struct servent** **result*, **char** **buffer*, **int** **buflen*)		NULL
struct servent *getservbyport_r(**int** *port*, **const char** **proto*, **struct servent** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct servent *getservent_r(**struct hosten** **result*, **char** **buffer*, **int** *buflen*)		NULL
struct netent *getnetbyname_r(**const char** **name*, **struct netent** **result*,		NULL

char **buffer*, **int** *buflen*)	
struct netent ****getnetbyaddr_r**(**long** *net*, **int** *type*, **struct netent** ***result*, **char** ***buffer*, **int** *buflen*)	NULL
struct netent ****getnetent_r**(**struct netent** ***result*, **char** ***buffer*, **int** *buflen*)	NULL
struct protoent ****getprotobyname_r**(**const char** ***name*, **struct protent** ***result*, **char** ***buffer*, **int** *buflen*)	NULL
struct protoent ****getprotobynumber_r**(**int** *proto*, **struct protent** ***result*, **char** ***buffer*, **int** *buflen*)	NULL
struct protoent ****getprotoent_r**(**struct protent** ***result*, **char** ***buffer*, **int** *buflen*)	NULL

関数	ヘッダーファイル: poll.h	戻り値
int poll(**struct pollfd** **fds*, **unsigned int** *nfds*, **int** *timeout*)		– 1

関数	ヘッダーファイル: pwd.h	戻り値
struct passwd ****getpwent**( )		NULL
struct passwd ****getpwnam**(**const char** ***name*)		NULL
struct passwd ****getpwuid**(**uid_t** *uid*)		NULL
struct passwd ****putpwent**(**const struct passwd** ***p*, **FILE** ***f*)		– 1

関数	ヘッダーファイル: regex.h	戻り値
char ****regcomp**(**regex_t** ***preg*, **const char** ***pattern*, **int** *cflags*)		– 1
int regexec(**const regex_t** ***preg*, **const char** ***string*, **size_t** *nmatch*, **regmatch_t** *pmatch* [])		– 1
regerror(**int** *errcode*, **const regex_t** ***preg*, **char** ***errbuf*, **size_t** *eflags*)		– 1
void regfree(**regex_t** ***preg*)		N/A

関数	ヘッダーファイル: signal.h	戻り値
int kill(**pid_t** *pid*, **int** *sig*)		– 1
int sigaction(**int** *sig*, **const struct sigaction** ***act*, **struct sigaction** ***oact*)		– 1
int sigemptyset(**sigset_t** ***set*)		– 1
int sigfillset(**sigset_t** ***set*)		– 1

int sigaddset(**sigset_t** **set*, **int** *signo*)	– 1
int sigdelset(**sigset_t** **set*, **int** *signo*)	– 1
int sigismember(**const sigset_t** **set*, **int** *signo*)	– 1
int sigpending(**sigset_t** **set*)	– 1
int sigprocmask(**int** *how*, **const sigset_t** **set*, **sigset_t** **oset*)	– 1
int sigsuspend(**const sigset_t** **set*)	– 1
int gsignal(**int** *sig*)	– 1
int sighold(**int** *sig*)	– 1
int sigrelse(**int** *sig*)	– 1
int sigignore(**int** *sig*)	– 1
int sigpause(**int** *sig*)	– 1
int sigaltstack(**const stack_t** **ss*, **stack_t** **oss*)	– 1

関数	ヘッダーファイル: stdio.h	戻り値
FILE *fdopen(int fildes, **const char** **type*)		NULL

関数	ヘッダーファイル: stdlib.h	戻り値
char * **initstate(** **unsigned int** *seed*, **char** * *state*, **size_t** *size*)		NULL
long random()		0
char * **setstate(** **const char** *state*)		NULL
void srandom(**unsigned int** **seed*)		N/A
int mkstemp(char * tempplate)		-1

関数	ヘッダーファイル: stropts.h	戻り値
int fattach(**int** *fildes*, **const char** **path*)		– 1
int fdetach(**const char** **path*)		– 1
int getmsg(**int** *fildes*, **struct strbuf** **ctptr*, **struct strbuf** **dataptr*, **long** **flagsp*)		– 1
int getpmsg(**int** *fildes*, **struct strbuf** **ctptr*, **struct strbuf** **dataptr*, **int** **band*, **long** **flagsp*)		– 1
int isastream(**int** *fildes*)		– 1
int putmsg(**int** *fildes*, **struct strbuf** **ctptr*, **struct strbuf** **dataptr*, **long** *flags*)		– 1
int putpmsg(**int** *fildes*, **struct strbuf** **ctptr*, **struct strbuf** **dataptr*, **int** *band*, **long** *flags*)		– 1

関数	ヘッダーファイル: sys/stat.h	戻り値
int mknod(**const char** **path*, **mode_t** *mode*, **dev_t** *dev*)		– 1
int fchmod(**int** *fildes*, **mode_t** *mode*)		– 1
int mkfifo(**const char** **path*, **mode_t** *mode*)		– 1

関数	ヘッダーファイル: sys/acct.h	戻り値
int acct(**const char** **path*)		– 1

関数	ヘッダーファイル: sys/fcntl.h	戻り値
int fcntl(**int** *fildes*, **int** *cmd*, .../* arg */)		– 1

関数	ヘッダーファイル: sys/msg.h	戻り値
int msgctl(**int** *msqid*, **int** *cmd*, **struct msqid_ds *buf)**		– 1
int msgget(**key_t** *key*, **int** *msgflg*)		– 1

関数	ヘッダーファイル: sys/mman.h	戻り値
void *mmap(**void** * *addr*, **size_t** *len*, **int** *prot*, **int** *flags*, **int** *fildes*, **off_t** *off*)		(void *) – 1
int mprotect(**void** **addr*, **size_t** *len*, **int** *port*)		– 1
int munmap(**void** **addr*, **size_t** *len*)		– 1

関数	ヘッダーファイル: sys/procset.h	戻り値
int sigsend(**idtype_t** *idtype*, **id_t** *id*, **int** *sig*)		– 1
int sigsendset(**procset_t** **psp*, **int** *sig*)		– 1

関数	ヘッダーファイル: sys/resource.h	戻り値
int getpriority(**int** *which*, **int** *who*)		– 1
int setpriority(**int** *which*, **int** *who*, **int** *prio*)		– 1
int getrlimit(**int** *resource*, **struct rlimit** **rlp*)		– 1
int setrlimit(**int** *resource*, **const struct rlimit** **rlp*)		– 1

関数	ヘッダーファイル: sys/socket.h	戻り値
int recvmsg(**int** *s*, **struct msghdr** **msg*, **int** *flags*)		– 1
int sendmsg(**int** *s*, **struct msghdr** **msg*, **int** *flags*)		– 1
int socketpair(**int** *domain*, **int** *type*, **int** *protocol*, **int** *sv*)		– 1

関数	ヘッダーファイル: sys/stat.h	戻り値
int mknod(const char **path*, **mode_t** *mode*, **dev_t** *dev*)		– 1
int fchmod(**int** *fildes*,**mode_t** *mode*)		– 1
int mkfifo(**const char** **path*, **mode_t** *mode*)		– 1

関数	ヘッダーファイル: sys/times.h	戻り値
clock_t times (**struct tms** **buffer*)		– 1

関数	ヘッダーファイル: sys/uio.h	戻り値
int writev(int *fildes*, **const struct iovec** **iov*, **int** *iovcnt*)		– 1

関数	ヘッダーファイル: sys/utsname.h	戻り値
int uname(struct utsname **name*)		– 1

関数	ヘッダーファイル: sys/wait.h	戻り値
pid_t wait (**int** *stat_loc)		0
pid_t waitpid(**pid_t** *pid*, **int** *stat_loc*, **int** *options*)		– 1

関数	ヘッダーファイル: syslog.h	戻り値
void syslog(**int** *priority*, **const char** *message, ...)		N/A
void openlog(**const char** *ident, **int** *logopt*, **int** *facility*)		N/A
void closelog(**void**)		N/A
int setlogmask(**int** *maskpri*)		– 1

関数	ヘッダーファイル: termios.h	戻り値
int cfgetispeed(**struct struct termios** *termios_p*)		NULL
spped_t cfgetospeed(**struct struct termios** *termios_p*)		NULL
int cfsetispeed(**struct termios** *termios_p*, **speed_t** *speed*)		– 1
int cfsetospeed(**struct termios** *termios_p*, **speed_t** *speed*)		– 1
int 1**tcdrain**(**int** *fildes*)		– 1
int tcflow(**int** *fildes*, **int** *action*)		– 1
int tcflush(**int** *fildes*, **int** *queue_selector*)		– 1
int tcgetattr(**int** *fildes*, **struct termios** *termios_p*)		– 1
int tcsendbreak(**int** *fildes*, **int** *duration*)		– 1
int tcsetattr(**int** *fildes*, **int** *optional_actions*, **const struct termios** *termios_p*)		– 1

関数	ヘッダーファイル: time.h	戻り値
int ascftime(**char** *s, **const char** *format, **const struct tm** *timeptr)		0
int cftime(**char** *s, **char** *format, **const time_t** *clock)		0
char ***ctime_r**(**const time_t** *clock, **char** *buf, **int** *buflen*)		NULL
struct tm ***localtime_r**(**const time_t** *clock, **struct tm** *res*)		NULL
struct tm ***gmtime_r**(**const time_t** *clock, **struct tm** *res*)		NULL
char ***asctime_r**(**const struct tm** *tm, **char** *buf, **int** *buflen*)		NULL
int clock_settime(**clockid_t** *clock_id*, **const struct timespec** *tp*)		– 1
int clock_gettime(**clockid_t** *clock_id*, **const struct timespec** *tp*)		– 1
int clock_getres(**clockid_t** *clock_id*, **const struct timespec** *res*)		– 1

int timer_create(**clockid_t** *clock_id*, **struct sigevent** **evp*, **timer_t** **timerid*)	– 1
int timer_delete(**timer_t** *timerid*)	– 1
int timer_settime(**timer_t** *timerid*, **int** *flags*, **const struct itimerspec** **value*, **struct itimerspec** **ovalue*)	– 1
int timer_gettime(**timer_t** *timerid*, **struct timespec** **value*)	– 1
int timer_getoverrun(**timer_t** *timerid*)	– 1
int nanosleep(**const struct timespec** **rqtp*, **struct timespec** **rmtp*)	– 1

関数	ヘッダーファイル: unistd.h	戻り値
int acct(**const char** **path*)		– 1
unsigned alarm(**unsigned int** *sec*)		0
int chown(**const char** **path*, **uid_t** *owner*, **gid_t** *group*)		– 1
int chroot(**const char** **path*)		– 1
char *ctermid(**char** **s*)		NULL
char *cuserid(**char** **s*)		NULL
int fchdir(**int** *fildes*)		– 1
int fchown(**int** *fildes*, **uid_t** *owner*, **gid_t** *group*)		– 1
int fchroot(**int** *fildes*)		– 1
int fsync(**int** *fildes*)		– 1
int ftruncate(**int** *fildes*, **off_t** *length*)		– 1
uid_t getuid(**void**)		– 1
uid_t geteuid(**void**)		– 1
gid_t getgid(**void**)		– 1
gid_t getegid(**void**)		– 1
int getgroups(**int** *gidsetsize*, **gid_t** **grouplist*)		– 1
gethostid(**void**)		0
char *getlogin(**void**)		NULL
char *getlogin_r(**char** **name*, **int** *namelen*)		NULL
pid_t getpgrp(**void**)		NULL
pid_t getppid(**void**)		NULL
pid_t getpgid(**pid_t** *pid*)		– 1
char *gettxt(**const char** **msgid*, **const char** **dflt_str*)		NULL
pid_t getsid(**pid_t** *pid*)		NULL
pid_t setsid(**void**)		NULL
int link(**const char** **existing*, **const char** **newfile*)		– 1
int lchown(**const char** **path*, **uid_t** *owner*, **gid_t** *group*)		– 1
int mincore(**caddr_t** *addr*, **size_t** *len*, **char** **vec*)		– 1
int nice(**int** *incr*)		– 1
long pathconf(**const char** **path*, **int** *name*)		– 1

int pause(void)	– 1
void profil(unsigned short **buff*, **unsigned int** *bufsiz*, **unsigned int** *offset*, **unsigned int** *scale*)	N/A
int ptrace(int *request*, **pid_t** *pid*, **int** *addr*, **int** *data*)	– 1
int readlink(const char **path*, **void** **buf*, **size_t** *bufsiz*)	– 1
void *sbrk(int *incr*)	(void *) – 1
int setuid(uid_t *uid*)	– 1
int setegid(gid_t *egid*)	– 1
int seteuid(uid_t *euid*)	– 1
int setgid(gid_t *gid*)	– 1
int setgroups(int *ngroups*, **const gid_t** **grouplist*)	– 1
int setpgid(pid_t *pid*, **pid_t** *pgid*)	– 1
pid_t setpgrp(void)	– 1
int stime(const time_t **tp*)	– 1
int symlink(const char **name1*, **const char** **name2*)	– 1
void sync(void)	N/A
long sysconf(int *name*)	– 1
pid_t tcgetpgrp(int *fildes*)	NULL
int tcsetpgrp(int *fildes*, **pid_t** *pgid*)	– 1
int truncate(const char **path*, **off_t** *length*)	– 1
char ttyname(int *fildes*)	NULL
char ttyname_r(int *fildes*, **char** **buf*, **int** *len*)	NULL
void vhangup(void)	N/A

関数	ヘッダーファイル: wchar.h	戻り値
int fwide(FILE **stream*, **int** *c*)		0
int mbsinit(const mbstate_t **ps*)		0
wchar_t wmemchr(const wchar_t **str1*, **wchar_t** *c*, **size_t** *count*)		NULL
int wmemcmp(const wchar_t **str1*, **const wchar_t** **str2*, **size_t** *count*)		NULL
int wcwidth(const wchar_t **c*)		– 1
int wcswidth(const wchar_t **str1*, **size_t** *width*)		– 1

関数	ヘッダーファイル: wctype.h	戻り値
wint_t towctrans(win_t *wc*, **wctrans_t** *desc*)		– 1
wctrans_t wctrans(const char **prop*)		0
wctype_t wctype(const char **prop*)		0

索引

__bool_true_false_are_defined, 762
_IOFBF, 771
_IOLBF, 771
_IONBF, 771
_wfopen(), 952, **955**

abort(), 842, **844**
abs(), 842, **845**
acct(), 1073, 1076
acos(), 295, **298**
acosh(), 295, **299**
aio.h, iv
aio_cancel(), 1063, 1068
aio_error(), 1063, 1068
aio_fsync(), 1064, 1068
aio_read(), 1063, 1068
aio_return(), 1063, 1068
aio_suspend(), 1063, 1068
aio_write(), 1063, 1068
alarm(), 1076
array.h, iv
arraycopy(), 740, **741**
arrow(), 4, *see* CPlot
ascftime(), 1062, 1064, 1075
asctime(), 934, **936**
asctime_r(), 1064, 1075
asin(), 295, **300**
asinh(), 295, **301**
assert(), 1, **2**
assert.h, iv
atan(), **302**
atan2(), 295, **303**
atanh(), **305**
atexit(), 842, **846**
atoc(), 842, **848**
atof(), 842, **850**
atoi(), 842, **851**
atol(), 842, **851**
atoll(), 842, **851**
autoScale(), 4, *see* CPlot
axes(), 5, *see* CPlot
axis(), 4, *see* CPlot
axisRange(), 4, *see* CPlot

balance(), 367, **372**
barSizd(), 5
barSize(), *see* CPlot
bindtextdomain(), 1061, 1068
boo, 762
border(), 5, *see* CPlot
borderOffsets(), 5, *see* CPlot
boundingBoxOrigin(), 5
boxBorder(), 5, *see* CPlot
boxFill(), 5, *see* CPlot
boxWidth(), 5, *see* CPlot
bsearch(), 842, **853**
btowc, **956**
btowc(), 952, 1062, 1067
BUFSIZ, 771

cabs(), 237, **239**
cacos(), 237, **240**
cacosh(), 237, **241**
calloc(), 842, **855**
carg(), 237, **242**
casin(), 237, **243**
casinh(), 237, **244**
catan(), 237, **245**
catanh(), 237, **246**
Cauchy, 367, **672**
cbrt(), 295, **306**, 1069
ccompanionmatrix(), 367, **375**
ccos(), 237, **247**
ccosh(), 237, **248**
cdeterminant(), 367, **376**
cdiagonal(), 367, **378**
cdiagonalmatrix(), 367, **381**
ceil(), 295, **307**
cexp(), 237, **249**
cfevalarray(), 367, **383**
cfgetispeed(), 1075
cfgetospeed(), 1075
cfsetispeed(), 1075
cfsetospeed(), 1075
cftime(), 1062, 1064, 1075
cfunm(), 367, **385**
CH_CARRAY, 739
CH_CARRAYPTR, 739
CH_CARRAYPTRTYPE, 747
CH_CARRAYTYPE, 747
CH_CARRAYVLA, 739
CH_CARRAYVLATYPE, 747

CH_CHARRAY, 739
CH_CHARRAYPTRTYPE, 739, 747
CH_CHARRAYTYPE, 747
CH_CHARRAYVLATYPE, 739, 747
CH_UNDEFINETYPE, 747
changeViewAngle(), 5, *see* CPlot
CHAR_BIT, 281
CHAR_MAX, 281
CHAR_MIN, 281
charpolycoef(), 367, **387**
Chebyshev, 367
ChebyshevVandermonde, **674**
chinfo, 227
chinfo(), 227, **229**
choldecomp(), 367, **389**
Chow, 367, **675**
chown(), 1076
chplot.h, iv
chroot(), 1076
chshell.h, iv
ChType_t, 738
cimag(), 237, **250**
cinverse(), 367, **393**
circle(), 5, *see* CPlot
Circul, 367, **676**
clearerr(), 770, **773**
Clement, 367, **677**
clinsolve(), 367, **395**
clock(), 934, **938**
clock_getres(), 1064, 1075
clock_gettime(), 1064, 1075
clock_settime(), 1064, 1075
CLOCKS_PER_SEC, 933
clog(), 237, **251**
closedir(), 1068
closelog(), 1075
cmean(), 367, **397**
colorBox(), 5, *see* CPlot
combination(), 367, **399**
companionmatrix(), 367, **400**
complex.h, iv
complexsolve(), 367, **401**
complexsolvePP(), 367, **407**
complexsolvePR(), 367, **409**
complexsolveRP(), 367, **411**
complexsolveRR(), 367, **413**
complexsolveRRz(), 367, **416**
condnum(), 367, **418**
conj(), 237, **252**
contourLabel(), 5, *see* CPlot
contourLevels(), 5, *see* CPlot
contourMode(), 5, *see* CPlot
conv(), 367, **420**
conv2(), 367, **425**
coordSystem(), 5, *see* CPlot
copyright, ii
copysign(), 295, **308**
corrcoef(), 367, **429**
correlation2(), 367, **432**
cos(), 295, **309**
cosh(), 295, **310**
covariance(), 367, **434**
cpio.h, iv
CPlot, **4**
  ~CPlot, 4
  arrow(), 4, **11**
  autoScale(), 4, **15**
  axes(), 5, **22**
  axis(), 4, **17**
  axisRange(), 4, **18**
  barSize(), 5, **24**
  border(), 5, **25**
  borderOffsets(), 5, **27**
  boundingBoxOrigin(), 5
  boxBorder(), 5, **28**
  boxFill(), 5, **29**
  boxWidth(), 5, **32**
  changeViewAngle(), 5, **33**
  circle(), 5, **35**
  colorBox(), 5, **37**
  contourLabel(), 5, **39**
  contourLevels(), 5, **41**
  contourMode(), 5, **44**
  coordSystem(), 5, **46**
  CPlot(), 4
  data(), 5, **51**
  data2D(), 5, **54**
  data2DCurve(), 5, **59**
  data3D(), 5, **60**
  data3DCurve(), 5, **68**
  data3DSurface(), 5, **70**
  dataFile(), 5, **73**
  dataSetNum(), 5, **77**
  deleteData(), 5, **78**
  deletePlots(), 5, **78**
  dimension(), 5, **79**
  displayTime(), 5, **80**
  enhanceText(), 5, **81**
  func2D(), 5, **86**
  func3D(), 5, **88**
  funcp2D(), 5, **89**
  funcp3D(), 5, **91**
  getLabel(), 5, **92**
  getOutputType(), 5, **93**
  getSubplot(), 5, **94**
  getTitle(), 5, **98**
  grid(), 6, **99**
  isUsed(), 6, **102**

label(), 6, **103**
legend(), 6, **104**
legendLocation(), 6, **106**
legendOption(), 6, **107**
line(), 6, **111**
lineType(), 6, **113**
margins(), 6, **117**
origin(), 6, **118**
outputType(), 6, **119**
plotting(), 6, **158**
plotType(), 6, **130**
point(), 6, **159**
pointType(), 6, **161**
polarPlot(), 6, **164**
polygon(), 6, **166**
rectangle(), 6, **169**
removeHiddenLine(), 6, **172**
scaleType(), 6, **174**
showMesh(), 6, **176**
size(), 6, **178**
size3D(), 6, **180**
sizeOutput(), 6, **181**
sizeRatio(), 6, **181**
smooth(), 6, **183**
subplot(), 6, **185**
text(), 6, **186**
tics(), 6, **187**
ticsDay(), 6, **188**
ticsDirection(), 6, **190**
ticsFormat(), 6, **191**
ticsLabel(), 6, **193**
ticsLevel(), 6, **195**
ticsLocation(), 6, **197**
ticsMirror(), 6, **199**
ticsMonth(), 6, **201**
ticsPosition(), 6, **202**
ticsRange(), 6, **203**
title(), 6, **204**
cpolyeval(), 367, **437**
cpow(), 237, **253**
cproduct(), 367, **439**
creal(), 237, **254**
cross(), 367, **441**
crypt.h, iv
csin(), 237, **255**
csinh(), 237, **256**
csqrt(), **257**
csum(), 367, **442**
ctan(), 237, **258**
ctanh(), 237, **259**
ctermid(), 1076
ctime(), 934, **940**
ctime_r(), 1064, 1075
ctrace(), 367, **444**
ctriangularmatrix(), 367, **446**
ctype.h, iv
cumprod(), 367, **449**
cumsum(), 367, **451**
curvefit(), 367, **453**
cuserid(), 1076

data(), 5, *see* CPlot
data2D(), 5, *see* CPlot
data2DCurve(), 5, *see* CPlot
data3D(), 5, *see* CPlot
data3DCurve(), 5, *see* CPlot
data3DSurface(), 5, *see* CPlot
dataFile(), 5, *see* CPlot
dataSetNum(), 5, *see* CPlot
DBL_DIG, 278
DBL_EPSILON, 280
DBL_MANT_DIG, 278
DBL_MAX, 279
DBL_MAX_10_EXP, 279
DBL_MAX_EXP, 279
DBL_MIN, 280
DBL_MIN_10_EXP, 279
DBL_MIN_EXP, 279
deconv(), 367, **457**
deleteData(), 5, *see* CPlot
deletePlots(), 5, *see* CPlot
DenavitHartenberg, 367, **679**
DenavitHartenberg2, 367, **681**
derivative(), 367, **460**
derivatives(), 367, **462**
determinant(), 367, **465**
dgettext(), 1061, 1068
diagonal(), 367, **467**
diagonalmatrix(), 367, **470**
difference(), 367, **472**
difftime(), 934, **941**
dimension(), 5, *see* CPlot
dirent.h, iv
displayTime(), 5, *see* CPlot
div(), 842, **856**
div_t, 841
dlfcn.h, iv
dot(), 367, **473**
Dramadah, 367, **683**

EDOM, 277
eigen(), 367, **474**
eigensystem(), 367, **479**
EILSEQ, 277
endhostent(), 1070
endnetent(), 1070
endprotoent(), 1070
endservent(), 1070
enhanceText(), 5, *see* CPlot

EOF, 771
ERANGE, 277
erf(), 295, **311**, 1069
erfc(), 295, **312**, 1069
errno.h, iv
exit(), 842, **857**
EXIT_FAILURE, 841
EXIT_SUCCESS, 841
exp(), 295, **313**
exp2(), 295, **314**
expm(), 367, **480**
expm1(), 295, **315**

fabs(), 295, **316**
factorial(), 367, **482**
false, 762
fat tach(), 1072
fattach(), 1065
fchdir(), 1076
fchmod(), 1073, 1074
fchown(), 1076
fchroot(), 1066, 1076
fclose(), 770, **774**
fcntl(), 1073
fcntl.h, iv
fdetach(), 1065, 1072
fdim(), 295, **317**
fdopen(), 1072
fenv.h, iv
feof(), 770, **775**
ferror(), 770, **776**
fevalarray(), 367, **483**
fflush(), 770, **777**
fft(), 367, **485**
fgetc(), 770, **778**
fgetgrent(), 1061
fgetgrent_r(), 1061
fgetpos(), 770, **779**
fgets(), 770, **781**
fgetwc(), 952, **958**
fgetws(), 952, **960**
Fiedler, 367, **685**
FILE, 772
FILENAME_MAX, 771
filpud(), 367
filter(), 367, **491**
filter2(), 367
findvalue(), 367, **498**
fliplr(), 367, **500**
flipud(), **502**
float.h, iv
floor(), 295, **318**
FLT_DIG, 278
FLT_EPSILON, 280
FLT_MANT_DIG, 278
FLT_MAX, 279
FLT_MAX_10_EXP, 279
FLT_MAX_EXP, 279
FLT_MIN, 280
FLT_MIN_10_EXP, 279
FLT_MIN_EXP, 279
FLT_RADIX, 278
fma(), 295, **319**
fmax(), 295, **320**
fmin(), 295, **321**
fminimum(), 367, **504**
fminimums(), 367, **508**
fmod(), 295, **322**
fopen(), 770, **782**
FOPEN_MAX, 771
fpclassify(), **323**, 1064, 1067, 1070
fplotxy(), 8, **206**
fplotxyz(), 9, **208**
fpos_t, 772
fprintf(), 770, **784**
fputc(), 770, **791**
fputs(), 770, **793**
fputwc(), 952, **962**
fputws(), 952, **964**
Frank, 367, **686**
fread(), 770, **794**
free(), 842, **859**
freopen(), 770, **795**
frexp(), 295, **324**
fscanf(), 770, **796**
fseek(), 770, **802**
fsetpos(), 770, **804**
fsolve(), 367, **511**
fsync(), 1076
ftell(), 770, **805**
ftruncate(), 1076
func2D(), 5, *see* CPlot
func3D(), 5, *see* CPlot
funcp2D(), 5, *see* CPlot
funcp3D(), 5, *see* CPlot
funm(), 367, **514**
fwide(), 952, **966**, 1062, 1067, 1077
fwrite(), 770, **807**
fzero(), 367, **516**

gamma(), 1069
gamma_r(), 1060, 1064, 1070
gcd(), 367, **518**
Gear, 367, **687**
getc(), 770, **810**
getchar(), 770, **809**
getegid(), 1076
getenv(), 842, **860**

geteuid(), 1076
getgid(), 1076
getgrent(), 1069
getgrent_r(), 1069
getgrgid_r(), 1061
getgrnam_r(), 1061
getgroups(), 1076
gethostbyaddr_r(), 1060, 1070
gethostbyname_r(), 1060, 1070
gethostent(), 1070
gethostent_r(), 1060, 1070
gethostid(), 1076
getLabel(), 5, *see* CPlot
getlogin(), 1076
getlogin()_r, 1076
getlogin_r(), 1061
getmsg(), 1065, 1072
getnetbyaddr(), 1070
getnetbyaddr_r(), 1060, 1071
getnetbyname(), 1070
getnetbyname_r(), 1060, 1070
getnetent(), 1070
getnetent_r(), 1060, 1071
getnum(), 367, **520**
getOutputType(), 5, *see* CPlot
getpgid(), 1076
getpgrp(), 1076
getpmsg(), 1065, 1072
getppid(), 1076
getpriority(), 1074
getprotobyname_r(), 1060, 1071
getprotobynumber_r(), 1060, 1071
getprotoent(), 1070
getprotoent_r(), 1060, 1071
getpwent(), 1071
getpwnam(), 1071
getpwuid(), 1071
getrlimit(), 1074
gets(), 770, **812**
getservbyname_r(), 1060, 1070
getservbyport_r(), 1060, 1070
getservent(), 1070
getservent_r(), 1060, 1070
getsid(), 1066, 1076
getSubplot(), 5, *see* CPlot
gettext(), 1061, 1068
getTitle(), 5, *see* CPlot
gettxt(), 1066, 1076
getuid(), 1076
getwc(), 952, **968**
getwchar(), 952, **970**
glob(), 1069
glob.h, iv
globfree(), 1069
gmtime(), 934, **942**
gmtime_r(), 1064, 1075
grid(), 6, *see* CPlot
grp.h, iv
gsignal(), 1072

Hadamard, 367, **689**
Hankel, 367, **690**
hessdecomp(), 367, **521**
Hilbert, 367, **691**
histogram(), 367, **524**
householdermatrix(), 367, **527**
HUGE_VAL, 296
hypot(), 295, **325**

identitymatrix(), 367, **529**
ifft(), 367, **530**
ilogb(), 295, **326**, 1069
inet.h, iv
inet_lnaof(), 1068
inet_makeaddr(), 1068
inet_netof(), 1068
inet_network(), 1068
INFINITY, 296
initgroups(), 1069
initstate(), 1072
INT_MAX, 282
INT_MIN, 281
integral1(), 367, **532**
integral2(), 367, **535**
integral3(), 367, **537**
integration2(), 367, **539**
integration3(), 367, **541**
interp1(), 367, **544**
interp2(), 367, **546**
inttypes.h, iv
inverse(), 367, **549**
InverseHilbert, 367, **692**
ioctlocket(), 1060, 1063, 1067
iostream.h, iv
isalnum(), 261, **262**
isalpha(), 261, **263**
isastream(), 1065, 1072
iscnan(), 237, **260**
iscntrl(), 261, **265**
iscnum(), 842, **861**
isdigit(), 261, **266**
isenv(), 842, **863**
isfinite(), 296, **327**
isgraph(), 261, **267**
isgreater(), 296, **328**
isgreaterequal(), 296, **329**
isinf(), 296, **330**
iskey, 227
iskey(), 227, **231**

isless(), 296, **331**
islessequal(), 296, **332**
islessgreater(), 296, **333**
islower(), 261, **268**
isnan(), 296, **334**
isnormal(), 297, **335**, 1060, 1064, 1067, 1070
isnum(), 842, **864**
iso646.h, iv
isprint(), 261, **270**
ispunct(), 261, **271**
isspace(), 261, **272**
isstudent, 227
isstudent(), 227, **233**
isunordered(), 297, **336**
isupper(), 261, **273**
isUsed(), 6, *see* CPlot
isvar, 227
isvar(), 227, **234**
iswalnum(), 1038, **1040**
iswalpha(), 1038, **1041**
iswcntrl(), 1038, **1042**
iswctype(), 1038, **1043**
iswdigit(), 1038, **1045**
iswgraph(), 1038, **1046**
iswlower(), 1038, **1047**
iswnum(), 842, **865**
iswprint(), 1038, **1048**
iswpunct(), 1038, **1049**
iswspace(), 1038, **1050**
iswupper(), 1038, **1051**
iswxdigit(), 1038, **1052**
isxdigit(), 261, **274**

jump_buf, 727

kill(), 1071

L_tmpnam, 771
label(), 6, *see* CPlot
labs(), 842, **845**
LC_ALL, 284
LC_COLLATE, 284
LC_CTYPE, 284
LC_MONETARY, 284
LC_NUMERIC, 284
LC_TIME, 284
lchown(), 1076
lcm(), 367, **551**
lconv, 285
LDBL_DIG, 278
LDBL_EPSILON, 280
LDBL_MANT_DIG, 278
LDBL_MAX, 279
LDBL_MAX_10_EXP, 279
LDBL_MAX_EXP, 279
LDBL_MIN, 280
LDBL_MIN_10_EXP, 279
LDBL_MIN_EXP, 279
ldexp(), 295, **337**
ldiv_t, 841
legend(), 6, *see* CPlot
legendLocation(), 6, *see* CPlot
legendOption(), 6, *see* CPlot
lgamma(), 295, **338**, 1069
lgamma_r(), 1070
libintl.h, iv
limits.h, iv
lindata(), 367, **553**
line(), 6, *see* CPlot
lineType(), 6, *see* CPlot
link(), 1076
linsolve(), 367, **555**
linspace(), 367, **557**
lio_listio(), 1063, 1068
LLONG_MAX, 282
LLONG_MIN, 282
llsqcovsolve(), 367, **560**
llsqnonnegsolve(), 367, **562**
llsqsolve(), 367, **564**
local_timer(), 1064
locale.h, iv
localeconv(), 284, **286**
localtime(), 934, **943**
localtime_r(), 1075
log(), 295, **339**
log10(), 295, **340**
log1p(), 295, **341**
log2(), 295, **342**
logb(), 295, **343**
logdata(), 367, **569**
logm(), 367, **567**
logspace(), 367, **571**
LONG_MAX, 282
LONG_MIN, 282
longjmp(), 726, **730**
lrint(), 295, **344**
lround(), 295, **345**
ludecomp(), 367, **573**

Magic, 367, **693**
malloc(), 842, **866**
malloc.h, iv
margins(), 6, *see* CPlot
math.h, iv
maxloc(), 367, **577**
maxv(), 367, **579**
MB_CUR_MAX, 841
MB_LEN_MAX, 281
mblen(), 842, **868**

mbrlen(), 952, **971**, 1062, 1067
mbrtow(), 1067
mbrtowc(), 952, **973**, 1062
mbsinit(), 952, **975**, 1062, 1067, 1077
mbsrtowcs(), 952, **977**, 1062, 1067
mbstate_t, 951
mbstowcs(), 842, **870**
mbtowc(), 842, **872**
mean(), 367, **580**
median(), 367, **583**
memchr(), 892, **894**
memcmp(), 892, **895**
memcpy(), 892, **896**
memmove(), 892, **897**
memset(), 892, **898**
mincore(), 1066, 1076
minloc(), 367, **585**
minv(), 367, **586**
mkfifo(), 1073, 1074
mknod(), 1073, 1074
mkstemp(), 1072
mktime(), 934, **944**
mlock(), 1065
mmap(), 1073
modf(), 296, **346**
mprotect(), 1073
mq_receive(), 1061
mqueue.h, iv
msgctl(), 1073
msgget(), 1073
msync(), 1065
munlock(), 1065
munmap(), 1073

NAN, 296
nan(), 296, **347**
nanosleep(), 1064, 1076
nearbyint(), 296, **348**
netconfig.h, iv
netdb.h, iv
netdir.h, iv
netinet/in.h, iv
new.h, iv
nextafter(), 296, **349**
nexttoward(), 296, **350**
nice(), 1076
norm(), 367, **587**
nullspace(), 367, **591**
numeric.h, iv

oderk(), 367, **594**
oderungekutta(), 367, **603**
odesolve(), 367, **605**
offsetof(), 763
opendir(), 1068
openlog(), 1075
origin(), 6, *see* CPlot
orthonormalbase(), 367, **607**
outputType(), 6, *see* CPlot

Pascal, 367, **694**
pathconf(), 1076
pause(), 1077
perror(), 770, **813**
pinverse(), 367, **610**
PLOT_ANGLE_DEG, **7**, 47, 165
PLOT_ANGLE_RAD, **7**, 47, 165
PLOT_AXIS_X, **7**, 15, 17, 18, 22, 92, 103, 175, 187, 189, 192, 194, 198, 199, 201–203
PLOT_AXIS_X2, **7**, 15, 17, 18, 22, 92, 103, 175, 187, 189, 192, 194, 198, 199, 201–203
PLOT_AXIS_XY, **7**, 16, 17, 19, 22, 103, 175, 187, 189, 192, 194, 198, 199, 201, 203, 204
PLOT_AXIS_XYZ, **7**, 16, 19, 23, 103, 175, 187, 189, 192, 194, 199, 201, 203, 204
PLOT_AXIS_Y, **7**, 15, 17, 18, 22, 93, 103, 175, 187, 189, 192, 194, 198, 199, 201–203
PLOT_AXIS_Y2, **7**, 15, 17, 19, 22, 93, 103, 175, 187, 189, 192, 194, 198, 199, 201, 203, 204
PLOT_AXIS_Z, **7**, 16, 19, 22, 93, 103, 175, 187, 189, 192, 194, 199, 201, 203, 204
PLOT_BORDER_ALL, **7**
PLOT_BORDER_BOTTOM, **7**
PLOT_BORDER_LEFT, **7**
PLOT_BORDER_RIGHT, **7**
PLOT_BORDER_TOP, **7**
PLOT_BOXFILL_EMPTY, **7**, 30
PLOT_BOXFILL_PATTERN, **7**, 30
PLOT_BOXFILL_SOLID, **7**, 30
PLOT_CONTOUR_BASE, **7**, 44
PLOT_CONTOUR_SURFACE, **7**, 44
PLOT_COORD_CARTESIAN, **7**, 47
PLOT_COORD_CYLINDRICAL, **7**, 47
PLOT_COORD_SPHERICAL, **7**, 47
PLOT_OFF, **7**, 16, 17, 25, 38, 39, 99, 172, 176, 187, 200
PLOT_ON, **7**, 16, 17, 25, 38, 39, 99, 172, 176, 187, 200
PLOT_OUTPUTTYPE_DISPLAY, **7**, 94, 119
PLOT_OUTPUTTYPE_FILE, **7**, 94, 120
PLOT_OUTPUTTYPE_STREAM, **7**, 94, 120
PLOT_PLOTTYPE_BOXERRORBARS, **7**, 29, 30, 32, 33, 133
PLOT_PLOTTYPE_BOXES, **7**, 29, 30, 32, 33, 132, 158
PLOT_PLOTTYPE_BOXXYERRORBARS, **7**, 29, 30, 33, 133
PLOT_PLOTTYPE_CANDLESTICKS, **7**, 29, 30, 33, 133, 150

PLOT_PLOTTYPE_DOTS, **7**, 132
PLOT_PLOTTYPE_FILLEDCURVES, **8**, 29, 30, 132, 154, 155, 157
PLOT_PLOTTYPE_FINANCEBARS, **8**, 24, 134, 151
PLOT_PLOTTYPE_FSTEPS, **8**, 132
PLOT_PLOTTYPE_HISTEPS, **8**, 132
PLOT_PLOTTYPE_IMPULSES, **8**, 132
PLOT_PLOTTYPE_LINES, **8**, 44, 132
PLOT_PLOTTYPE_LINESPOINTS, **8**, 132
PLOT_PLOTTYPE_POINTS, **8**, 132
PLOT_PLOTTYPE_STEPS, **8**, 132
PLOT_PLOTTYPE_SURFACES, **8**, 44, 132
PLOT_PLOTTYPE_VECTORS, **8**
PLOT_PLOTTYPE_Vectors, 132, 149
PLOT_PLOTTYPE_XERRORBARS, **8**, 24, 134
PLOT_PLOTTYPE_XERRORLINES, **8**, 135
PLOT_PLOTTYPE_XYERRORBARS, **8**, 24, 25, 134
PLOT_PLOTTYPE_XYERRORLINES, **8**, 135
PLOT_PLOTTYPE_YERRORBARS, **8**, 25, 134, 153
PLOT_PLOTTYPE_YERRORLINES, **8**, 135
PLOT_PLOTTYPES_VECTORS, 150
PLOT_SCALETYPE_LINEAR, **8**, 175
PLOT_SCALETYPE_LOG, **8**, 175
PLOT_TEXT_CENTER, **8**, 186
PLOT_TEXT_LEFT, **8**, 186
PLOT_TEXT_RIGHT, **8**, 186
PLOT_TICS_IN, **8**, 190
PLOT_TICS_OUT, **8**, 190
plotting(), 6, *see* CPlot
plotType(), 6, *see* CPlot
plotxy(), 9, **211**
plotxyf(), 9, **217**
plotxyz(), 9, **219**
plotxyzf(), 9, **224**
point(), 6, *see* CPlot
pointType(), 6, *see* CPlot
polarPlot(), 6, *see* CPlot
poll(), 1066, 1071
poll.h, iv
polycoef(), **615**
polyder(), 367, **618**
polyder2(), 367, **620**
polyeval(), 367, **623**
polyevalarray(), 367, **625**
polyevalm(), 367, **627**
polyfit(), 367, **629**
polygon(), 6, *see* CPlot
pow(), 296, **351**
printf(), 770, **815**
product(), 367, **633**
profil(), 1077
pthread.h, iv
ptrace(), 1077
ptrdiff_t, 764
putc(), 770, **817**
putchar(), 770, **816**
putmsg(), 1065, 1072
putpmsg(), 1065, 1072
putpwent(), 1071
puts(), 770, **819**
putwc(), 952, **979**
putwchar(), 952, **981**
pwd.h, iv

qrdecomp(), 367, **635**
qrdelete(), 367, **641**
qrinsert(), 367, **645**
qsort(), 842, **874**

raise(), **736**
rand(), 842, **876**
RAND_MAX, 841
random(), 1072
rank(), 367, **649**
rcmd(), 1070
rcondnum(), 367, **651**
re_comp.h, iv
readdir(), 1069
readdir_r(), 1069
readline.h, iv
readlink(), 1077
realloc(), 842, **877**
rectangle(), 6, *see* CPlot
recvmsg(), 1074
regcmp(), 1064, 1069
regcomp(), 1071
regerror(), 1071
regex(), 1064, 1069
regex.h, iv
regexec(), 1071
regfree(), 1071
remainder(), 296, **353**, 1069
remenv(), 842, **879**
remove(), 770, **820**
removeHiddenLine(), 6, *see* CPlot
remquo(), 296, **354**
rename(), 770, **821**
residue(), 367, **653**
rewind(), 770, **823**
rewinddir(), 1069
rexec(), 1070
rinlt(), **355**
rint(), 296, 1069
roots(), 367, **658**
Rosser, 367, **696**
rot90(), 367, **660**
round(), 296, **356**
rresvport(), 1070
rsf2csf(), 367, **662**

ruserok(), 1070

sbrk(), 1077
scalbn(), 296, 1060, 1069
scalbn() & scalbln(), **357**
scaleType(), 6, *see* CPlot
scanf(), 770, **824**
SCHAR_MAX, 281
SCHAR_MIN, 281
sched.h, iv
schurdecomp(), 367, **665**
SEEK_CUR, 771
SEEK_END, 771
SEEK_SET, 771
seekdir(), 1069
semaphore.h, iv
sendmsg(), 1074
setbuf(), 770, **826**
setegid(), 1061, 1077
seteuid(), 1061, 1077
setgid(), 1077
setgrent(), 1069
setgroups(), 1077
sethostent(), 1070
setjmp(), 726, **728**
setjmp.h, iv
setlocale(), 284, **292**
setlogmask(), 1075
setnetent(), 1070
setpgid(), 1077
setpgrp(), 1077
setpriority(), 1074
setprotoent(), 1070
setrlimit(), 1074
setservent(), 1070
setsid(), 1076
setstate(), 1072
setuid(), 1077
setvbuf(), 770, **827**
shmopen(), 1065
shmunlink(), 1065
showMesh(), 6, *see* CPlot
SHRT_MAX, 281
SHRT_MIN, 281
sig_atomic_t, 732
SIG_DFL, 732
SIG_ERR, 732
SIG_IGN, 732
sigaction(), 1071
sigaddset(), 1072
sigaltstack(), 1065, 1072
sigdelset(), 1072
sigemptyset(), 1071
sigfillset(), 1071
sighold(), 1065, 1072
sigignore(), 1065, 1072
sigismember(), 1072
sign(), 367, **668**
signal(), 732, **733**
signal.h, iv
signbit(), 297, **359**, 1060, 1064, 1067, 1070
sigpause(), 1072
sigpending(), 1072
sigprocmask(), 1072
sigrelse(), 1065, 1072
sigsend(), 1065, 1073
sigsendset(), 1061, 1065, 1073
sigsuspend(), 1072
sin(), 296, **360**
sinh(), 296, **361**
size(), 6, *see* CPlot
size_t, 764, 772
size3D(), 6, *see* CPlot
sizeofelement, 227
sizeofelement(), 227, **235**
sizeOutput(), 6, *see* CPlot
sizeRatio(), 6, *see* CPlot
smooth(), 6, *see* CPlot
snprintf(), 770, **829**
socketpair(), 1074
sort(), 367, **669**
specialmatrix(), 367, **672**
sprintf(), 770, **830**
sqrt(), 296, **362**
sqrtm(), 367, **701**
srand(), 842, **880**
srandom(), 1072
sscanf(), 770, **831**
std(), 367, **703**
stdarg.h, iv
stdbool.h, iv
stddef.h, iv
stderr, 771
stdio.h, iv
stdlib.h, iv
stdn, 771
stdout, 771
stime(), 1077
str2ascii(), **899**
str2mat(), **900**
stradd(), 892, **902**
strcasecmp, 892
strcasecmp(), **903**
strcat(), 892, **904**
strchr(), 892, **905**
strcmp(), 892, **906**
strcoll(), 892, **907**
strconcat(), 892, **908**

strcpy(), 892, **909**
strcspn(), 892, **910**
strdup(), 892, **911**
strerror(), 892, **912**
strftime(), 934, **946**
strgetc(), 892, **913**
string.h, iv
strjoin(), 893, **914**
strlen(), 893, **915**
strncasecmp(), 893, **916**
strncat(), 893, **918**
strncmp(), 893, **919**
strncpy(), 893, **920**
stropts.h, iv
strpbrk(), 893, **921**
strputc(), 893, **922**
strrchr(), 893, **923**
strrep(), 893, **924**
strspn(), 893, **926**
strstr(), 893, **927**
strtoascii(), 892
strtod(), 842, **882**
strtof(), **882**
strtok(), 893, **930**
strtok_r, 893, **928**
strtol(), 842, **885**
strtold(), **882**
strtoll(), 842, **885**
strtomat(), 892
strtoul(), 842, **885**
strtoull(), **885**
strxfrm(), 893, **932**
subplot(), 6, *see* CPlot
sum(), 367, **705**
svd(), 367, **707**
symlink(), 1077
sync(), 1077
sysconf(), 1077
syslog(), 1075
syslog.h, iv
system(), 842, **887**

tan(), 296, **363**
tanh(), 296, **364**
tar.h, iv
tcdrain(), 1075
tcflow(), 1075
tcflush(), 1075
tcgetattr(), 1075
tcgetpgrp(), 1077
tcsendbreak(), 1075
tcsetattr(), 1075
tcsetpgrp(), 1077
telldir(), 1069
termios.h, iv
text(), 6, *see* CPlot
textdomain(), 1061, 1068
tgamma(), 296, **365**
tgmath.h, iv
tics(), 6, *see* CPlot
ticsDay(), 6, *see* CPlot
ticsDirection(), 6, *see* CPlot
ticsFormat(), 6, *see* CPlot
ticsLabel(), 6, *see* CPlot
ticsLevel(), 6, *see* CPlot
ticsLocation(), 6, *see* CPlot
ticsMirror(), 6, *see* CPlot
ticsMonth(), 6, *see* CPlot
ticsPosition(), 6, *see* CPlot
ticsRange(), 6, *see* CPlot
time(), 934, **950**
time.h, iv
timer_create(), 1064, 1076
timer_delete(), 1064, 1076
timer_getoverrun(), 1064, 1076
timer_gettime(), 1064, 1076
timer_settime(), 1064, 1076
times(), 1074
title(), 6, *see* CPlot
tiuser.h, iv
tm, 933, 951
TMP_MAX, 771
tmpfile(), 771, **832**
tmpnam(), 771, **833**
Toeplitz, 367, **697**
tolower(), 261, **275**
toupper(), 261, **276**
towctrans(), 1038, **1053**, 1062, 1077
towlower(), 1038, **1055**
towupper(), 1038, **1056**
trace(), 367, **714**
triangularmatrix(), 367, **716**
true, 762
trunc(), 296, **366**
truncate(), 1077
ttyname(), 1077
ttyname_r(), 1061, 1077

UCHAR_MAX, 281
UINT_MAX, 282
ULLONG_MAX, 282
ULONG_MAX, 282
uname(), 1074
ungetc(), 771, **835**
ungetwc(), 952, **982**
unistd.h, iv
unwrap(), 367, **719**
urand(), 367, **721**

USHRT_MAX, 281
utime.h, iv

va_arg(), 737, **743**
va_arraydim(), 737, **744**
va_arrayextent(), 737, **745**
va_arraynum(), 737, **746**
va_arraytype(), 737, **747**
va_copy(), 737, **749**
va_count(), 738, **751**
va_datatype(), 738, **752**
va_dim(), 738, **753**
va_elementtype(), 738, **754**
va_end(), 738, **755**
va_extent(), 738, **756**
va_list(), 738
VA_NOARG, 737
va_start(), 738, **757**
va_tagname(), 738, **760**
vandermatrix(), 367
Vandermonde, 367, **698**
vfprintf(), 771, **837**
vhangup(), 1077
vprintf(), 771, **838**
vsnprintf(), 771, **839**
vsprintf(), 771, **840**

wait(), 1075
wait.h, iv
waitid(), 1066
waitpid(), 1075
wchar.h, iv
WCHAR_MAX, 951
WCHAR_MIN, 951
wchar_t, 764
wcrtomb(), 952, **984**, 1062, 1067
wcscat(), 952, **986**
wcschr(), 952, **988**
wcscmp(), 952, **990**
wcscoll(), 952, **992**
wcscpy(), 952, **994**
wcscspn(), 952, **995**
wcsftime(), 952, **997**
wcslen(), 952, **999**
wcsncat(), 953, **1000**
wcsncmp(), 953, **1002**
wcsncpy(), 953, **1004**
wcspbrk(), 953, **1006**
wcsrchr(), 953, **1008**
wcsrtombs(), 953, **1010**, 1062, 1067
wcsspn(), 953, **1012**
wcsstr(), 953, **1014**, 1062, 1067
wcstod(), 953, **1016**
wcstok(), 953, **1019**
wcstol(), 953, **1021**
wcstombs(), 842, **888**
wcswidth(), 1077
wcsxfrm(), 953, **1023**
wctob(), 953, **1025**, 1062, 1067
wctomb(), 842, **890**
wctrans(), 1038, **1057**, 1062, 1077
wctrans_t, 1037
wctype(), 1038, **1058**, 1077
wctype.h, iv
wctype_t, 1037
wcwidth(), 1077
WEOF, 951, 1037
Wilkinson, 367, **699**
wint_t, 951, 1037
wmemchr(), 953, **1027**, 1062, 1067, 1077
wmemcmp(), 953, **1029**, 1062, 1067, 1077
wmemcpy(), 953, **1031**, 1062, 1067
wmemmove(), 953, **1033**, 1062, 1067
wmemset(), 953, **1035**, 1062, 1067
writev(), 1074
WSAAsyncGetHostByAddr(), 1059, 1063, 1066
WSAAsyncGetHostByName(), 1059, 1063, 1066
WSAAsyncGetProtoByName(), 1059, 1063, 1066
WSAAsyncGetProtoByNumber(), 1059, 1063, 1066
WSAAsyncGetServByName(), 1059, 1063, 1066
WSAAsyncGetServByPort(), 1059, 1063, 1066
WSAAsyncSelect(), 1059, 1063, 1066
WSACancelAsyncRequest(), 1059, 1063, 1067
WSACancelBlockingCall(), 1059, 1063, 1067
WSACleanup(), 1059, 1063, 1067
WSAGetLastError(), 1059, 1063, 1067
WSAIsblocking(), 1059, 1063, 1067
WSASetBlockingHook(), 1060, 1063, 1067
WSASetLastError(), 1060, 1063, 1067
WSAStartup(), 1060, 1063, 1067
WSAUnhookBlockingHook(), 1060, 1063, 1067

xcorr(), 367, **723**

表記規則, iv