

CodeWarrior®

MSL C Reference



最新の情報については CodeWarrior の Release Notes フォルダをご覧ください。

Revised: 991202 rdl-JP000323

Metrowerks CodeWarrior © Copyright 1993-2000 by Metrowerks Inc. and its Licensors. All rights reserved.

お客様は、本 CD に記録されている文書を個人使用目的に限り、プリントすることができます。この場合を除いて、Metrowerks Inc. からの書面による承諾なしに、本 CD に記録されている文書の全部、または、一部をいかなる形態、方法（電子的、物理的な複製、または、写真複写、録音録画、その他すべての情報記録、再生システムを含む）により、複製または伝達することを禁じます。

Metrowerks の名称、ロゴ、CodeWarrior、PowerPlant、Metrowerks University は、Metrowerks Inc. の登録商標です。

Geekware、Discover Programming は、Metrowerks Inc. の商標です。

記載の商標および登録商標は、各社が保有します。

CD に記録されているすべてのソフトウェアおよび文書は、CodeWarrior QuickStart の巻末に記述されているライセンス契約が適用されます。

連絡先：

Japan	メトロワークス株式会社 150-0042 東京都渋谷区宇田川町 36-6 ワールド宇田川ビル 8F TEL : (03) 3780-6091 FAX : (03) 3780-6092
U.S.A.	Metrowerks Corporation 9801 Metric Boulevard, Suite 100 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
WWW サーバ	http://www.metrowerks.co.jp/ http://www.metrowerks.com/
ユーザー登録	j-register@metrowerks.com
テクニカルサポート	j-support@metrowerks.com
購入 / 契約更新	j-sales@metrowerks.com
インフォメーション	j-info@metrowerks.com

目次

第 1 章 はじめに	17
このマニュアルの構成	17
ANSI C の標準規格	18
ANSI C ライブラリと Mac OS	18
コンソール I/O と Mac OS	18
コンソール I/O と Windows	19
マニュアルの表記規則	19
表記について	19
互換性	20

第 2 章 <code>alloca.h</code>	21
<code>alloca.h</code> の概要	21
<code>alloca</code>	21

第 3 章 <code>assert.h</code>	23
<code>assert.h</code> の概要	23
<code>assert</code>	23

第 4 章 <code>console.h</code>	25
<code>console.h</code> の概要	25
<code>ccommand</code>	25
<code>clrscr</code>	27
<code>getch</code>	28
<code>InstallConsole</code>	28
<code>kbhit</code>	29
<code>ReadCharsFromConsole</code>	29
<code>RemoveConsole</code>	30
<code>__ttyname</code>	30
<code>WriteCharsToConsole</code>	31

第 5 章 <code>crtl.h</code>	33
<code>crtl.h</code> の概要	33
<code>Argc</code>	33
<code>Argv</code>	33
<code>_DlITerminate</code>	34
<code>environ</code>	34
<code>_HandleTable</code>	34
<code>_CRTStartup</code>	35
<code>_RunInit</code>	35

_SetupArgs	35
<hr/>	
第 6 章 ctype.h	37
ctype.h の概要	37
文字セットのサポート	37
isalnum	38
isalpha	39
iscntrl	40
isdigit	40
isgraph	41
islower	41
isprint	42
ispunct	42
isspace	43
isupper	43
isxdigit	44
tolower	45
toupper	46
<hr/>	
第 7 章 div_t.h	47
div_t.h の概要	47
div_t	47
ldiv_t	47
<hr/>	
第 8 章 errno.h	49
errno.h の概要	49
errno	49
<hr/>	
第 9 章 fcntl.h	53
fcntl.h の概要	53
fcntl.h と UNIX の互換性	53
creat	53
fcntl	54
open	56
umask	58
<hr/>	
第 10 章 float.h	61
float.h の概要	61
浮動小数点数の指数部	61

第 11 章 FSp_fopen.h	63
FSp_fopen.h の概要	63
FSp_fopen	63
第 12 章 io.h	65
io.h の概要	65
_chdir	65
_chdrive	66
_fileno	66
_get_osfhandle	66
_getcwd	67
GetHandle	67
_heapmin	68
_isatty	68
_makepath	69
_open_osfhandle	69
_searchenv	70
第 13 章 limits.h	71
limits.h の概要	71
整数型の限界	71
第 14 章 locale.h	73
locale.h の概要	73
地域設定情報の仕様	73
localeconv	74
setlocale	74
第 15 章 malloc.h	77
malloc.h の概要	77
alloca	77
第 16 章 math.h	79
math.h の概要	79
浮動小数点の計算	81
NaN (Not a Number)	81
浮動小数点エラーテスト	82
インラインを積極的に利用する	82
浮動小数点分類マクロ	82

列挙定数	82
fpclassify	82
isfinite	83
isnan	83
isnormal	84
signbit	84
浮動小数点 Math 関数	85
acos	85
acosf	85
acosl	85
asin	86
asinf	86
asinl	86
atan	86
atanf	87
atanl	87
atan2	87
atan2f	88
atan2l	88
ceil	88
ceilf	89
ceill	89
cos	89
cosf	90
cosl	90
cosh	90
coshf	91
coshl	91
exp	91
expf	92
expl	92
fabs	92
fabsf	93
fabsl	93
floor	93
floorf	94
floorl	94
fmod	94
fmodf	95
fmodl	95
frexp 実装	95
frexpf	96
frexpl	97

isgreater.	97
isgreaterless.	97
isless.	98
islessequal.	98
isunordered.	98
ldexp.	99
ldexpf.	100
ldexpl.	100
log.	100
logf.	101
logl.	101
log10.	101
log10f.	102
log10l.	102
modf.	102
modff.	103
modfl.	103
pow.	103
powf.	104
powl.	104
sin.	104
sinf.	105
sinl.	105
sinh.	105
sinhf.	106
sinhl.	106
sqrt.	106
sqrtf.	107
sqrtl.	107
tan.	108
tanf.	108
tanl.	109
tanh.	109
tanhf.	109
tanhl.	110
HUGE_VAL.	110
C9X の実装.	110
acosh.	110
asinh.	111
atanh.	111
copysign.	111
erf.	112
erfc.	112

exp2	113
expm1	113
fdim	114
fmax	114
fmin	114
gamma	115
hypot.	115
lgamma	116
log1p.	116
log2	117
logb	117
nan	118
nearbyint	118
nextafter	119
remainder.	119
remquo	120
rint	120
rinttol	121
round.	121
roundtol	122
scalb	122
trunc	123

第 17 章 path2fss.h 125

path2fss.h の概要	125
path2fss	125

第 18 章 Process.h 127

Process.h の概要	127
_beginthreadex	127
_endthreadex	128

第 19 章 setjmp.h 129

setjmp.h の概要	129
非ローカルジャンプと例外の取り扱い	129
longjmp	129
setjmp	130

第 20 章 signal.h 133

signal.h の概要	133
------------------------	-----

シグナル処理	133
signal	135
raise	137
Be 特有のシグナルの取り扱い	137
sigaction	139
sigprocmask	139
sigpending	140
sigsuspend	140
kill	141
send_signal	141
struct vregs	141
 第 21 章 SIOUX と WinSIOUX	143
SIOUX と WinSIOUX の概要	143
SIOUX と WinSIOUX を使う	143
Windows 用 WinSIOUX	144
WinSIOUX を使うプロジェクトを作成	144
WinSIOUX のカスタマイズ	144
WinSIOUXclrscr	145
clrscr	146
Mac OS 用 SIOUX	146
SIOUX を用いたプロジェクトの作成	147
SIOUX のカスタマイズ	148
サイズと位置の変更	151
ユーザーアプリケーションでの SIOUX ウィンドウの使用	152
SIOUXclrscr	153
SIOUXHandleOneEvent	154
SIOUXSetTitle	155
 第 22 章 stat.h	157
stat.h の概要	157
stat.h と UNIX の互換性	157
Stat 構造体と定義	157
fstat	158
mkdir	160
stat	161
 第 23 章 stdarg.h	163
stdarg.h の概要	163
関数の可変引数	163
va_arg	163

va_end	164
va_start	164
<hr/>	
第 24 章 stddef.h	167
stddef.h の概要	167
共通に利用される定義	167
NULL	167
offsetof	167
ptrdiff_t	167
size_t	168
wchar_t	168
<hr/>	
第 25 章 stdio.h	169
stdio.h の概要.	169
標準入力 / 出力	170
ストリーム	170
ファイル位置指標	171
ファイルの終わりとエラー	171
拡張文字とバイト文字のストリーム方向.	171
ストリーム方向と標準の入力 / 出力.	172
clearerr	172
fclose	173
fdopen	175
feof	176
ferror.	177
fflush.	179
fgetc	180
fgetpos	181
fgets	183
fopen.	184
fprintf	187
fputc	191
fputs	193
fread	194
freopen	196
fscanf	197
fseek	201
fsetpos	204
ftell	204
fwrite	205
getc	206
getchar	208

gets	209
perror	210
printf	211
putc	217
putchar	218
puts	219
remove	220
rename	221
rewind	222
scanf	224
setbuf	228
setvbuf	229
sprintf	231
sscanf	232
tmpfile	233
tmpnam	234
ungetc	235
vfprintf	237
vprintf	239
vsprintf	240

第 26 章 stdlib.h 243

stdlib.h の概要	243
abort	244
abs	245
atexit	246
atof	248
atoi	249
atol	249
bsearch	250
calloc	253
div	255
exit	256
free	257
getenv	258
labs	259
ldiv	260
malloc	260
mblen	261
mbstowcs	261
mbtowc	262
qsort	263

rand	264
realloc	265
srand	266
strtod	266
strtol	268
strtoul	269
system	270
vec_calloc	270
vec_free	271
vec_malloc	271
vec_realloc	271
wcstombs	272
wctomb	273

第 27 章 string.h 275

string.h の概要	275
memchr	276
memcmp	278
memcpy	279
memmove	279
memset	280
strcasecmp	281
strcat	281
strchr	282
strcmp	283
strcoll	284
strcpy	285
strcspn	286
strdup	287
strerror	288
_stricmp	288
strlen	289
strncasecmp	290
strncat	290
strncmp	291
strncpy	292
_strnicmp	293
strpbrk	294
strrchr	295
_strrev	296
strspn	296
strstr	297

strtok	298
strxfrm	300
_strupr	301

第 28 章 time.h 303

time.h の概要	303
日付と時間	303
struct tm	304
tzname	305
asctime	305
clock	306
ctime	307
difftime	308
gmtime	309
localtime	310
mktime	310
_strdate	311
strftime	311
time	316
tzset	317

第 29 章 unistd.h 319

unistd.h の概要	319
unistd.h と UNIX の互換性	319
chdir	320
close	322
cuserid	324
exec	325
getcwd	327
getlogin	328
getpid	328
isatty	330
lseek	331
read	332
rmdir	333
sleep	333
ttyname	334
unlink	335
write	336

第 30 章	unix.h	339
	unix.h の概要	339
	unix.h と UNIX の互換性	339
	大域変数	339
	_fcreator	339
	_ftype	340
	fdopen	341
	fileno.	342
	tell	343
第 31 章	utime.h	345
	utime.h の概要	345
	utime.h と UNIX の互換性	345
	utime.	345
	utimes	347
第 32 章	utsname.h	349
	utsname.h の概要.	349
	utsname.h と UNIX の互換性	349
	uname	349
第 33 章	wchar.h	351
	wchar.h の概要	351
	拡張文字とバイト文字のストリーム方向.	353
	ストリーム方向と標準の入力 / 出力	353
	定義	353
	fgetwc	353
	fgetws	354
	fwprintf	354
	fputwc	355
	fputws	356
	fwscanf	356
	getwc	357
	getwchar	358
	putwc	358
	putwchar	359
	swprintf	359
	swscanf	360
	towctrans	360
	__vwscanf	361

__vswscanf	361
vswscanf	362
vfwprintf	362
vswprintf	363
vwprintf	364
wasctime	364
watof	365
wscat	365
wcschr	366
wcscmp	366
wscoll	367
wcscspn	367
wcscpy	368
wcslen	368
wcsncat	369
wcsncmp	369
wcsncpy	370
wcspbrk	370
wcsspn	371
wcsrchr	371
wcsstr	372
wctod	372
wctok	373
wcsftime	373
wcsxfrm	374
wctime	374
wctrans	375
wmemchr	375
wmemcmp	376
wmemcpy	376
wmemmove	377
wmemset	378
wprintf	378
wscanf	379

第 34 章 wctype.h	381
wctype.h の概要	381
iswalnum	381
iswalp	382
iswcntrl	382
iswdigit	383
iswgraph	383

iswlower	384
iswprint	384
iswpunct	384
iswspace	385
iswupper	385
iswxdigit	386
towlower	386
towupper	387

索引	389
----	-----

第 1 章 はじめに

このリファレンスマニュアルでは ANSI ライブラリと CodeWarrior の C 言語で用いられる拡張ライブラリについて説明します。

このマニュアルの構成

C のヘッダファイルは、アルファベット順に並んでいます。ヘッダファイルの項目もアルファベット順に記載されています。可能な限り、それぞれの関数の使用例をサンプルコードとして添付しています。

[「alloca.h の概要」\(p21\)](#) : スタックからの動的メモリ割り当てに関する非 ANSI の `alloca()` 関数などについて

[「assert.h の概要」\(p23\)](#) : `assert()` マクロを取り扱う ANSI C の例外処理について

[「console.h の概要」\(p25\)](#) : Mac OS のコンソール用ルーチンについて

[「crtl.h の概要」\(p33\)](#) : Win32 のコンソール用ルーチンについて

[「ctype.h の概要」\(p37\)](#) : ANSI の文字に関する機能について

[「div_t.h の概要」\(p47\)](#) : 2 つの配列に関する `math` ルーチンについて

[「errno.h の概要」\(p49\)](#) : ANSI の広域エラー変数について

[「fcntl.h の概要」\(p53\)](#) : 非 ANSI のファイルのコントロールについて

[「float.h の概要」\(p61\)](#) : ANSI の浮動小数点型の制限について

[「FSp_fopen.h の概要」\(p63\)](#) : Mac OS ファイルのオープン用ルーチンについて

[「io.h の概要」\(p65\)](#) : Windows に共通の入出カストリーム用のルーチンについて

[「limits.h の概要」\(p71\)](#) : ANSI の整数型の制限について

[「locale.h の概要」\(p73\)](#) : ANSI の文字セット、数値と通貨の書式について

[「malloc.h の概要」\(p77\)](#) : Windows 用の `alloca` 関数について

[「math.h の概要」\(p79\)](#) : ANSI の浮動小数点に関する数学 (math) 用の機能について

[「path2fss.h の概要」\(p125\)](#) : Mac OS の拡張ファイルルーチンについて

[「Process.h の概要」\(p127\)](#) : Windows のスレッドプロセス用ルーチンについて

[「setjmp.h の概要」\(p129\)](#) : ANSI のプロセス状態の保存と復帰に利用される方法について

[「signal.h の概要」\(p133\)](#) : ANSI のソフトウェア割り込みの仕様について

[「SIOUX と WinSIOUX の概要」\(p143\)](#) : Metrowerks SIOUX と WinSIOUX コンソール用のエミュレーションについて

[「stat.h の概要」\(p157\)](#) : 非 ANSI のファイル統計とその機能について

[「stdarg.h の概要」\(p163\)](#) : ANSI のカスタム用の変数引数に関する機能について

[「stddef.h の概要」\(p167\)](#) : ANSI の標準規格の定義について

[「stdio.h の概要」\(p169\)](#) : ANSI の標準入出力用のルーチンについて

[「stdlib.h の概要」\(p243\)](#) : ANSI の共通ライブラリに関する機能について

[「string.h の概要」\(p275\)](#) : ANSI の `null` で終わる文字配列に関する機能について

[「time.h の概要」\(p303\)](#) : ANSI の `clock`、`date`、`time` の変換とその機能について

[「unistd.h の概要」\(p319\)](#) : 共通の非 ANSI の機能について

[「unix.h の概要」\(p339\)](#) : Metrowerks の非 ANSI の機能について

[「utime.h の概要」\(p345\)](#) : 非 ANSI のファイルアクセス時間に関する機能について

[「utsname.h の概要」\(p349\)](#) : 非 ANSI の装置の名前付けに関する機能について

[「wchar.h の概要」\(p351\)](#) : 単一、および配列の拡張文字セットについて

[「wctype.h の概要」\(p381\)](#) : 拡張文字セットのタイプ比較機能について

ANSI C の標準規格

ANSI C の標準規格のライブラリは、「ANSI:Programming Language C / X3.159:1989」の仕様に準拠した Metrowerks CodeWarrior を用いて組み込まれています。このライブラリの関数、変数やマクロ変数は、C と C++ の両方のプログラムから利用することができます。

`unix.h`、`Aunistd.h`、`stat.h`、`fcntl.h`、`utsname.h` では、ANSI の標準規格ではありませんが UNIX システムと共通の関数がいくつか宣言されています。

ANSI C ライブラリと Mac OS

ANSI C ライブラリの関数のいくつかは、Mac OS 環境では完全には動作しません。Mac OS コンピュータのグラフィカルユーザーインターフェースの代わりに、文字ベースのユーザーインターフェースが用いられているためです。これらの関数が利用可能であっても、期待した通りには動作しないかもしれません。ANSI C の標準規格と Metrowerks の実装との矛盾は、各関数の項目で説明します。

ANSI C ライブラリは、Pascal ではなく C の文字列を使用しています。

コンソール I/O と Mac OS

ANSI 規格のライブラリでは、対話型のコンソール I/O (標準入力、標準エラー出力、標準出力) は常時オープンしていると仮定されています。このライブラリの関数の多くは、Mac OS コンピュータのグラフィカルなユーザーインターフェースではなく、文字指向のユーザーインターフェース上で利用するように設計されています。

`console.h` では、`ccommand()` が宣言されています。これは、コマンドライン引数を入力するためダイアログを表示します。

`SIoux.h` は `SIoux` パッケージの一部で、非プログラム式端末や TTY のようなウィンドウを生成します。プログラムが `stdin()`、`stdout()`、`stderr()`、`cin()`、`cout()`、あるいは `cerr()` を参照するときは必ずこのウィンドウを利用します。

コンソール I/O と Windows

ANSI 規格のライブラリでは、対話形のコンソール I/O (標準入力、標準エラー出力、標準出力) は常時オープンしていると仮定されています。このコマンドラインのインタフェースは、Windows95 と Windows NT のコンソールアプリケーションによって提供されます。特別な Windows コンソール用ルーチンに関しては、`io.h`、`crt1.h`、`process.h` のヘッダを調べてください。

マニュアルの表記規則

ここではマニュアルの表記規則を説明します。

表記について

特定の情報を表すスタイルについて説明します。

注意、警告、ヒント、および初心者用のヒント

「注意」は、重要な事実を言い換えたり、自明ではない事実に注意を向けます。

「警告」は、実行すると取り返しのつかないものを注意したり、発生する可能性のあるエラーを知らせます。

「ヒント」は、CodeWarrior をより活用するためのヒントです。

「初心者」は、プログラミングの初心者が用語や概念をよりよく理解できるようにします。

書体の規則

異なる書体 (Courier という書体です) のテキストは、ファイル名やフォルダ名、コード、またはコンピュータのハードディスク上で見られるその他の項目を示します。

CodeWarrior のメニューコマンドやオプション名などは括弧付き ([Post Linker] オプションなど) で示します。ターゲットによって名前が変更されるコマンドの変更部分は、斜体で表示します ([Target Settings])。ユーザーご本人が入力するテキストを、異なる書体 (Courier という書体です) で表示します (「Target」)。

下線付きの青色のテキストは (例: 『[IDE User Guide の概要](#)』(p17)) オンラインドキュメント (Adobe Acrobat など) でのハイパーテキストを示します。これをクリックすると該当ページへジャンプします。

互換性

それぞれの標準規格の関数の記述では、オペレーティングシステムやプロセッサでの関数の互換性に関する表があります。以下に互換性の表のサンプルを示します。

互換性 この関数は以下のターゲットと互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

互換性のあるターゲットは黒色のテキストで表示されます。

互換性のないターゲットは灰色で表示されます。

表中の空白の欄は、将来、ターゲットとしてサポートする予定です。

ANSI は「American National Standards Institute Programming Language C / X3.159.1989」を表します。

EMB/RTOS は Embedded Real Time Operating Systems を表します。現在、以下をサポートしています。

- PPC EABI (Embedded Application Binary Interface) を使用した PowerPC 組み込みプロセッサ
- Sony PlayStation オペレーティングシステム

Mac OS は、PowerPC あるいは 68K プロセッサで動作する Mac OS オペレーティングシステムを表します。

PalmOS は、3Com PalmPilot オペレーティングシステムを表します。

Win32 は、x86 で動作する Windows95 と WindowsNT オペレーティングシステムを表します。

CodeWarrior for Palm OS には C ライブラリのバイナリコードは含まれていません。ライブラリのソースとヘッダは説明のために表示します。

Inside CodeWarrior シリーズの書籍を読む場合、書籍が印刷された後に新しいターゲットが追加されていることもあるので注意してください。

ターゲットの情報は印刷された文書には掲載されない場合があるので、特別な関数がターゲットに対して互換性があるかどうかを調べるには、商品に添付されたオンラインドキュメントあるいはリリースノートをお読みください。

第 2 章 `alloca.h`

このヘッダファイルは 1 個の関数、[`alloca`](#) を定義します。これはスタックを使って高速にメモリの割り当てを行います。

`alloca.h` の概要

[「`alloca`」\(p21\)](#): スタックからメモリを割り当てる

`alloca`

スタック上で高速にメモリを割り当てます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

注意： Win32 ヘッダに関する `alloca()` 関数は、`malloc.h` において定義されています。

プロトタイプ

```
#include <alloca.h>
void *alloca(size_t nbytes);
```

引数 この関数の引数です。

`nbytes` `size_t` 割り当てのバイト数

注意点 この関数は、`nbyte` 長のメモリブロックのポインタを返します。ブロックは呼び出した関数のスタック上に作られます。関数が終了すると自動的にメモリも解放されます。

多くのメモリを割り当てようとする場合、設定パネルでプロジェクトのスタックサイズを増やしてください。

注意： `alloca` 関数はエンベデッド /RTOS システムには適用できません。リリースノートをお読みください。

戻り値 `alloca()` が成功した場合、メモリブロックのポインタを返します。エラーが発生した場合、`alloca()` は `NULL` を返します。

参照 [「calloc」\(p253\)](#)、[「free」\(p257\)](#)、[「malloc」\(p260\)](#)、[「realloc」\(p265\)](#)

第 3 章 assert.h

assert.h ヘッダファイルは、マクロやアサートを提供します。これらは診断メッセージを出力し、テストが失敗した場合プログラムを中止します。

assert.h の概要

assert.h はデバッグマクロを提供します。

[「assert」\(p23\)](#): 診断メッセージを出力し、テストが失敗した場合プログラムを中止する

assert

テストが失敗した場合、プログラムを中止します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <assert.h>`
`void assert(int expression);`

引数 この関数の引数です。

expression int boolean 式が評価されている。

注意点 式が失敗すると `assert()` マクロは `stderr` に診断メッセージを出力し、`abort()` を呼び出します。以下は診断メッセージの書式です。

```
file: line test -- assertion failed
abort -- terminating
```

意味を説明します。

```
file: ソースファイル
line: 行番号
test: 失敗した式
```

`assert()` を無効にするには、`#define NDEBUG (デバッグしない)` を `#include <assert.h>` の前に置いてください。

参照 [「abort」\(p244\)](#)

例 3.1 assert() の使用例

```
#undef NDEBUG
/* Make sure that assert() is enabled */
#include <assert.h>
#include <stdio.h>

int main(void)
{
    int x = 100, y = 5;
    printf("assert test.\n");

    /*This assert will output a message and abort the program */
    assert(x > 1000);
    printf("This will not execute if NDEBUG is undefined\n");
    return 0;
}
/* Output:
assert test.
foo.c:12 x > 1000 -- assertion failed
abort -- terminating
*/
```

第 4 章 console.h

console.h ヘッダファイルには関数 [ccommand](#) が含まれています。これはコマンドライン引数を用いるプログラムを移植するときに役立ちます。

console.h の概要

console.h ヘッダファイルに含まれる関数です。

[「ccommand」\(p25\)](#): コマンドライン引数を用いるプログラムを移植するための関数

[「clrscr」\(p27\)](#): ウィンドウをクリアし、バッファをフラッシュする

[「getch」\(p28\)](#): acii キーが押された場合、そのキーボード文字を返す

[「InstallConsole」\(p28\)](#): コンソールパッケージをインストールする

[「kbhit」\(p29\)](#): 任意のキーが押された場合、キーを検索しないで真を返す

[「ReadCharsFromConsole」\(p29\)](#): コンソールからバッファへ読み込む

[「RemoveConsole」\(p30\)](#): コンソールパッケージを削除する

[「_ttyname」\(p30\)](#): ファイル id に関連した端末名を返す (unix.h 関数 ttyname はこの関数を呼び出す)

[「WriteCharsToConsole」\(p31\)](#): コンソールウィンドウへの出力ストリームへ書き込む

注意: UNIX あるいは DOS プログラムを移植している場合、その他の UNIX 互換性のあるヘッダを必要とするかもしれません。

ccommand

SIUNIX プログラムに対して、コマンドライン引数を入力させます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <console.h>
int ccommand(char ***argv);
```

引数 この関数の引数です。

argv char *** コマンドラインにある第 2 のパラメータのアドレス

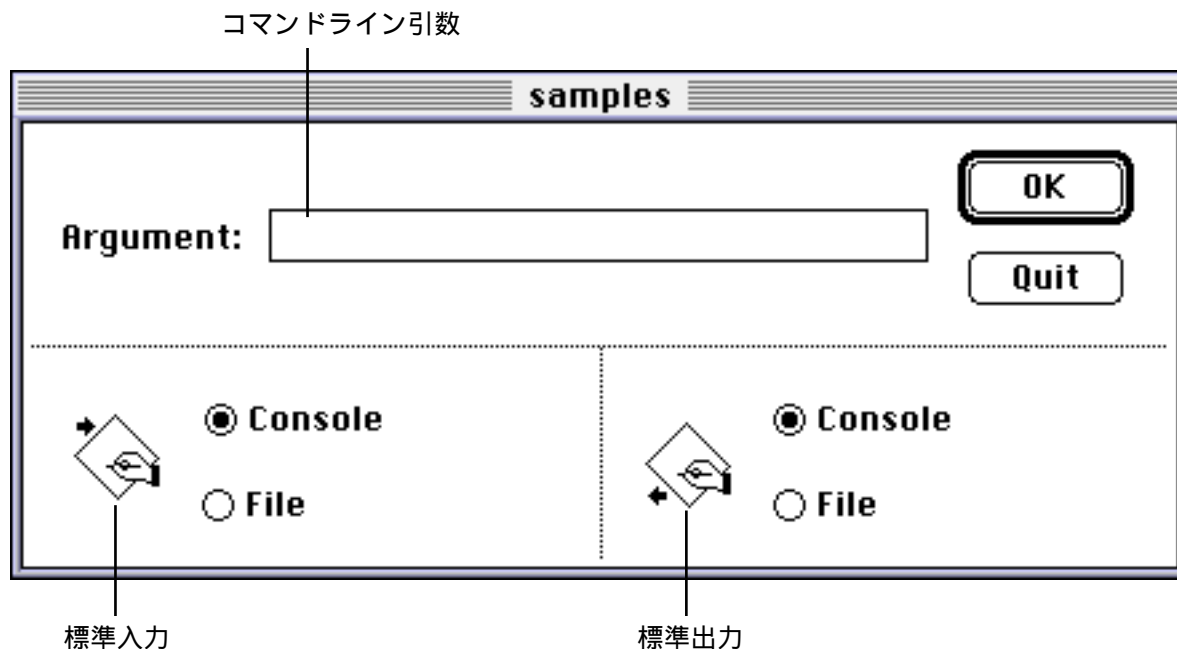
警告！ 関数 `ccommand()` はプログラムで生成する第 1 のコードでなくてはなりません。
`main` 関数の変数宣言の直後に置いてください。

注意点 この関数は「[コマンドダイアログ](#)」(p26) で示すように、引数を入力させて標準入出力をリダイレクトするダイアログを表示します。SIOUX をカスタマイズする情報やコンソールのオプションの設定については、「[SIOUX と WinSIOUX の概要](#)」(p143) を参照してください。

注意： `tdin`、`stdout`、`cin`、`cout` のみリダイレクトされます。標準のエラー通知、`stderr`、`cerr`、`clog` はリダイレクトされません。

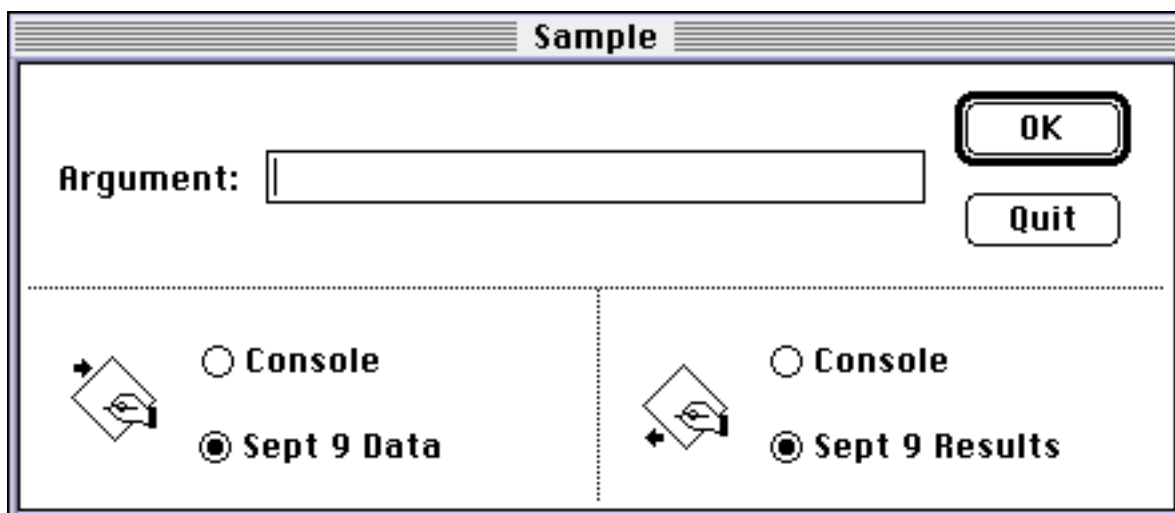
入力可能な引数の最大数は、`ccommand.c` で定義される `MAX_ARGS` 値によって決定され、25 に設定されています。この値を超える引数は無視されます。

図 4.1 コマンドダイアログ



引数のフィールドにコマンドライン引数を入力します。フィールドの下にあるボタンを使ってプログラムの標準入出力の方向を選択します。左側のボタンは標準入力用で、右側のボタンは標準出力用です。[Console] を選択した場合、プログラムは SIOUX ウィンドウに対して読み書きを行います。[File] を選択したとき、`ccommand()` は、読み書きするファイルを選択するための標準のファイルダイアログを表示します。ファイルを選択した後、[図 4.2 \(p27\)](#) のように、[File] という単語が選択したファイル名に置換されます。

図 4.2 入出力をファイルヘリダイレクト



関数 `ccommand()` は整数値を返し、文字配列へのポインタである引数を持ちます。それはダイアログに入力された引数の配列で、関数自体は入力された引数の数を返します。UNIX あるいは DOS では、最初の引数は 0 番目の引数でプログラムの名前です。[「`ccommand\(\)` の使用例」\(p27\)](#) ではコマンドラインの例について説明しています。

戻り値 入力された引数の数を返します。

参照 [「`SIIOUX` のカスタマイズ」\(p148\)](#)

例 4.1 `ccommand()` の使用例

```
#include <stdio.h>
#include <console.h>

int main(int argc, char *argv[])
{
    int i;

    argc = ccommand(&argv);

    for (i = 0; i < argc; i++)
        printf("%d. %s\n", i, argv[i]);
    return 0;
}
```

`clrscr`

コンソールウィンドウをクリアし、バッファをフラッシュします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <console.h>
void clrscr(void);
```

引数 なし。

注意点 この関数はスクリーンとバッファのすべてを選択してクリアします。Mac OS では `SIIOUX.h` から `SIIOUXclrscr` を呼び出します。Windows では `WinSIIOUX.h` から `WinSIIOUXclrscr` を呼び出します。

参照 [「SIIOUXclrscr」\(p153\)](#)

getch

ascii キーが押下されたときのそのキーボード文字を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <console.h>
int getch(void);
```

引数 なし。

注意点 この関数は、即時入力にコンソールのスタイルメニューで選択されたものを使います。

戻り値 ascii キーが押下された時に、そのキーボード文字を返します。

参照 [「kbhit」\(p29\)](#)

InstallConsole

コンソールパッケージをインストールします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <console.h>
extern short InstallConsole(short fd);
```

引数 この関数の引数です。

fd short 標準 I/O のファイルの説明

注意点 標準ストリームの 1 つに書き込みや、読み込みをする前にこの関数を呼び出してください。

戻り値 任意のエラーを返します。

参照 [「SIOUX のカスタマイズ」\(p148 \)](#) [「RemoveConsole」\(p30 \)](#)

kbhit

キーボードのキーが押下されると真を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <console.h>
int kbhit(void);
```

引数 なし。

注意点 キーボードのキーが押されると、キーを検索することなく真を返します。任意のキーが押されることによって、ループを停止させるときに利用します。

戻り値 キーボードのキーが押されたら、非ゼロを返します。

参照 [「getch」\(p28 \)](#)

ReadCharsFromConsole

コンソールからバッファへ読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <console.h>
extern long ReadCharsFromConsole
(char *buffer, long n);
```

引数 この関数の引数です。

buffer char * ストリームバッファ
n long 読み取る char の数

注意点 コンソールからバッファへ読み込みます。この関数は、read によって呼び出されます。

戻り値 任意のエラーを返します。

参照 [「WriteCharsToConsole」\(p31 \)](#)

RemoveConsole

コンソールパッケージを削除します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <console.h>
extern char *__ttyname(long fildes);
```

引数 この関数の引数です。

fildes	long	ファイルの説明
--------	------	---------

注意点 他の全てのストリームをクローズさせ、exit 関数 (atexit あるいは __atexit によってインストールされたもの) が呼ばれた後、呼び出されます。

戻り値 エラーから復旧する方法がないため、この関数は何も返す必要がありません。

参照 [「SIOUX のカスタマイズ」\(p148 \)](#) [「InstallConsole」\(p28 \)](#)

__ttyname

ファイル id に関連した端末の名前を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <console.h>
extern char *__ttyname(long fildes);
```

引数 この関数の引数です。

fildes	long	ファイルの説明
--------	------	---------

注意点 unix.h 関数 ttyname は、この関数を呼び出します(int 型のサイズと不一致のため、int 型から long 型へ移し替える必要がある)。

戻り値 ファイル `id` に関連した端末の名前を返します。

参照 [「ttyname」\(p334 \)](#)

WriteCharsToConsole

コンソールウィンドウの出力ストリームへ書き込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <console.h>
extern long WriteCharsToConsole
(char *buffer, long n);
```

引数 この関数の引数です。

buffer	char *	ストリームバッファ
n	long	書き込む char の数

注意点 この関数は `write` によって呼び出されます。

戻り値 任意のエラーを返します。

参照 [「ReadCharsFromConsole」\(p29 \)](#)

第 5 章 crt1.h

crt1.h ヘッダファイルは、Win32 x86 ターゲットに関係するさまざまなランタイムの宣言から構成されます。

crt1.h の概要

crt1.h ヘッダファイルは以下の機能を定義します。

- [「Argc」\(p33\)](#): 引数の個数
- [「Argv」\(p33\)](#): 引数リストの各引数
- [「_DllTerminate」\(p34\)](#): DLL が終了コードを実行させているかどうかを示す
- [「environ」\(p34\)](#): 環境ポインタ
- [「_HandleTable」\(p34\)](#): それぞれのファイルハンドラに割り当てられた構造体
- [「_CRTStartup」\(p35\)](#): C ランタイム開始ルーチンを初期化する
- [「_RunInit」\(p35\)](#): ランタイム、static クラスと変数の初期化を行う
- [「_SetupArgs」\(p35\)](#): コマンドライン引数の設定を行う

Argc

引数の個数です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <crt1.h>
extern int __argc;
```

注意点 コマンドラインの引数の個数として利用されます。

Argv

引数リストの各引数です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <crt1.h>
extern char **__argv;
```

注意点 コマンドラインの各引数を取り出します。

_DllTerminate

DLL が終了コードを実行しているかどうかを判定するフラグです。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <crt1.h>
extern int _DllTerminate;
```

注意点 このフラグは、DLL が終了コードを実行している時に設定されます。

environ

環境ポインタです。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <crt1.h>
extern char *(*environ);
```

注意点 これは環境へのポインタです。

_HandleTable

FileStruct は、それぞれのファイルハンドルが割り当てられた構造体です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <crt1.h>
typedef struct
```

```
{
    void *handle;
    char translate;
    char append;
} FileStruct;

extern FileStruct *_HandleTable[NUM_HANDLES];
extern int _HandPtr;
```

戻り値 変数 `_HandPtr` は、ハンドルのテーブルへのポインタです。

変数 `NUM_HANDLES` リストは可能なハンドル数です。

`_CRTStartup`

関数 `_CRTStartup` は、C ランタイム開始ルーチンです。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <crt1.h>`
`extern void _CRTStartup();`

注意点 なし。

`_RunInit`

関数 `_RunInit` はランタイム、全ての `static` クラス、変数を初期化します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <crt1.h>`
`extern void _RunInit();`

注意点 なし。

`_SetupArgs`

関数 `_SetupArgs` はコマンドライン引数を設定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <crt1.h>`
`extern void _SetupArgs();`

注意点 なし。

第 6 章 ctype.h

ctype.h ヘッダファイルは、文字タイプのテストと操作に関するマクロと関数を提供します。

ctype.h の概要

ctype.h ヘッダファイルは文字タイプのテストやアルファベットを大文字や小文字に変換するマクロを提供します。ctype.h マクロは ASCII 文字 (0x00 から 0x7F まで) と EOF の値をサポートします。これらのマクロは、Apple の Mac OS 拡張文字セット (0x80 から 0xFF まで) に関しては規定されていません。

ctype.h ヘッダは、文字タイプのテストに関するいくつかの関数を宣言しています。その関数を以下に示します。

[「isalnum」\(p38\)](#): アルファベットと数字をテストする

[「isalpha」\(p39\)](#): アルファベットをテストする

[「iscntrl」\(p40\)](#): 制御文字をテストする

[「isdigit」\(p40\)](#): 数字文字をテストする

[「isgraph」\(p41\)](#): グラフィック文字をテストする

[「islower」\(p41\)](#): 小文字をテストする

[「isprint」\(p42\)](#): 印刷可能な文字をテストする

[「ispunct」\(p42\)](#): 句読点文字をテストする

[「isspace」\(p43\)](#): 空白文字をテストする

[「isupper」\(p43\)](#): 大文字をテストする

[「isxdigit」\(p44\)](#): 16 進数文字をテストする

[「tolower」\(p45\)](#): 大文字から小文字に変換する

[「toupper」\(p46\)](#): 小文字から大文字に変換する

文字セットのサポート

MSL (Metrowerks Standard Library) の文字は ASCII 文字セットをテストします。拡張文字セットのテストは定義されていないので、システムによって動作しないことがあります。戻り値については [「文字テスト関数」\(p38\)](#) を参照してください。

表 6.1 文字テスト関数

関数	真を返す場合の c の値
isalnum(c)	アルファベットと数字の組み合わせ：[a-z], [A-Z], [0-9]
isalpha(c)	アルファベット：[a-z], [A-Z]
iscntrl(c)	削除文字 (0x7F) あるいは 0x00 から 0x1F の通常の制御文字
isdigit(c)	数字文字：[0-9]
isgraph(c)	空白を除いた感嘆符 (0x21) からチルド (0x7E) までの印刷用文字
islower(c)	小文字：[a-z]
isprint(c)	空白 (0x20) からチルドまでの印刷用文字
ispunct(c)	句読点文字 (制御文字やアルファベット、数字文字のいずれでもない文字)
isspace(c)	空白、タブ、改行、復帰改行、垂直タブ、書式送り
isupper(c)	大文字：[A-Z]
isxdigit(c)	16 進数文字 [0-9]、[A-F]、[a-f]

isalnum

文字の型を判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <ctype.h>
int isalnum(int c);
```

引数 この関数の引数です。

c int 評価されている文字

注意点 このマクロは c の整数値によって、真であれば非ゼロを返し、偽であればゼロを返します。
使用例は「[文字テスト関数の使用例](#)」(p39)を参照してください。

戻り値 [表 6.1](#) で文字テスト関数の戻り値について説明しています。

参照 「[tolower](#)」(p45)、 「[toupper](#)」(p46)

例 6.1 文字テスト関数の使用例

```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
    int a = 'F', b = '6', c = '#', d = 9;

    printf("isalnum for %c: %d\n", b, isalnum(b));
    printf("isalpha for %c: %d\n", a, isalpha(a));
    printf("iscntrl for %c: %d\n", d, iscntrl(d));
    printf("isdigit for %c: %d\n", d, isdigit(d));
    printf("isgraph for %c: %d\n", d, isgraph(d));
    printf("islower for %c: %d\n", a, islower(a));
    printf("isprint for %c: %d\n", d, isprint(d));
    printf("ispunct for %c: %d\n", c, ispunct(c));
    printf("isspace for %c: %d\n", d, isspace(d));
    printf("isupper for %c: %d\n", b, isupper(b));
    printf("isxdigit for %c: %d\n", a, isxdigit(a));

    return 0;
}
```

Output:

```
isalnum for 6: 32
isalpha for F: 2
iscntrl for : 64
isdigit for : 0
isgraph for : 0
islower for F: 0
isprint for : 0
ispunct for #: 8
isspace for : 64
isupper for 6: 0
isxdigit for F: 1
```

isalpha

文字の型を判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <ctype.h>`
`int isalpha(int c);`

引数 この関数の引数です。

c int 評価されている文字

注意点 このマクロは c の整数値によって、真であれば非ゼロ、偽であればゼロを返します。

戻り値 「[文字テスト関数](#)」(p38) で、文字テスト関数が何を返すかを説明しています。

例 6.2 使用例

使用例については、「[文字テスト関数の使用例](#)」(p39) を参照してください。

isctrl

文字の型を判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <ctype.h>
int isctrl(int c);

引数 この関数の引数です。

c int 評価されている文字

注意点 このマクロは c の整数値によって、真であれば非ゼロ、偽であればゼロを返します

戻り値 「[文字テスト関数](#)」(p38) では、文字テスト関数が何を返すかを説明しています。

例 6.3 使用例

使用例については、「[文字テスト関数の使用例](#)」(p39) を参照してください。

isdigit

文字の型を判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <ctype.h>


```
int isdigit(int c);
```

引数 この関数の引数です。

c int 評価されている文字

注意点 このマクロは c の整数値によって、真であれば非ゼロ、偽であればゼロを返します。

戻り値 [「文字テスト関数」\(p38\)](#) では、文字テスト関数が何を返すかを説明しています。

例 6.4 使用例

使用例については、[「文字テスト関数の使用例」\(p39\)](#) を参照してください。

isgraph

文字の型を判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <ctype.h>
int isgraph(int c);
```

引数 この関数の引数です。

c int 評価されている文字

注意点 このマクロは c の整数値によって、真であれば非ゼロ、偽であればゼロを返します。

戻り値 [「文字テスト関数」\(p38\)](#) では、文字テスト関数が何を返すかを説明しています。

例 6.5 使用例

使用例については、[「文字テスト関数の使用例」\(p39\)](#) を参照してください。

islower

文字の型を判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <ctype.h>`
`int islower(int c);`

引数 この関数の引数です。

`c` `int` 評価されている文字

注意点 このマクロは `c` の整数値によって、真であれば非ゼロ、偽であればゼロを返します。

戻り値 「[文字テスト関数](#)」(p38) では、文字テスト関数が何を返すかを説明しています。

例 6.6 使用例

使用例については、「[文字テスト関数の使用例](#)」(p39) を参照してください。

isprint

文字の型を判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <ctype.h>`
`int isprint(int c);`

引数 この関数の引数です。

`c` `int` 評価されている文字

注意点 このマクロは `c` の整数値によって、真であれば非ゼロ、偽であればゼロを返します。

戻り値 「[文字テスト関数](#)」(p38) では、文字テスト関数が何を返すかを説明しています。

例 6.7 使用例

使用例については、「[文字テスト関数の使用例](#)」(p39) を参照してください。

ispunct

文字の型を判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <ctype.h>`
`int ispunct(int c);`

引数 この関数の引数です。

`c` `int` 評価されている文字

注意点 このマクロは `c` の整数値によって、真であれば非ゼロ、偽であればゼロを返します。

戻り値 [「文字テスト関数」\(p38\)](#) では、文字テスト関数の戻り値を説明しています。

例 6.8 使用例

使用例については、[「文字テスト関数の使用例」\(p39\)](#) を参照してください。

isspace

文字の型を判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <ctype.h>`
`int isspace(int c);`

引数 この関数の引数です。

`c` `int` 評価されている文字

注意点 このマクロは `c` の整数値によって、真であれば非ゼロ、偽であればゼロを返します。

戻り値 [「文字テスト関数」\(p38\)](#) では、文字テスト関数が何を返すかを説明しています。

例 6.9 使用例

使用例については、[「文字テスト関数の使用例」\(p39\)](#) を参照してください。

isupper

文字の型を判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <ctype.h>
int isupper(int c);
```

引数 この関数の引数です。

c int 評価されている文字

注意点 このマクロは c の整数値によって、真であれば非ゼロ、偽であればゼロを返します。

戻り値 「[文字テスト関数](#)」(p38) では、文字テスト関数が何を返すかを説明しています。

例 6.10 使用例

使用例については、[「文字テスト関数の使用例」](#)(p39) を参照してください。

isxdigit

16 進の型を調べます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <ctype.h>
int isxdigit(int c);
```

引数 この関数の引数です。

c int 評価されている文字

注意点 このマクロは c の整数値によって、真であれば非ゼロ、偽であればゼロを返します。使用例に関しては「[文字テスト関数の使用例](#)」(p39) を参照してください。

戻り値 「[文字テスト関数](#)」(p38) では、文字テスト関数が何を返すかを説明しています。

例 6.11 使用例

使用例については、[「文字テスト関数の使用例」](#)(p39) を参照してください。

tolower

文字変換マクロです。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <ctype.h>`
`int tolower(int c);`

引数 この関数の引数です。

c int 評価されている文字

注意点 `tolower()` マクロは 1 文字の大文字を同じ小文字へ変換します。大文字ではない文字は何も変更せず値を返します。使用例は [「tolower\(\)、toupper\(\) の使用例」\(p45\)](#) を参照してください。

戻り値 `tolower()` は大文字と同じ小文字を返し、他の文字は変更せずに返します。

参照 [「isalpha」\(p39\)](#)、[「toupper」\(p46\)](#)

例 6.12 tolower()、toupper() の使用例

```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
    static char s[] =
        "*** DELICIOUS! lovely? delightful ***";
    int i;

    for (i = 0; s[i]; i++)
        putchar(tolower(s[i]));
    putchar('\n');

    for (i = 0; s[i]; i++)
        putchar(toupper(s[i]));
    putchar('\n');

    return 0;
}
```

Output:
** delicious! lovely? delightful **
** DELICIOUS! LOVELY? DELIGHTFUL **

toupper

文字変換マクロです。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <ctype.h>`
`int toupper(int c);`

引数 この関数の引数です。

c int 評価されている文字

注意点 `toupper()` マクロは小文字を同じ大文字に変換し、他の文字は変更せずに返します。

戻り値 `toupper()` は小文字と同じ大文字を返し、他の文字は変更せずに返します。

参照 [「isalpha」\(p39\)](#)、[「tolower」\(p45\)](#)

例 6.13 使用例

[「tolower\(\)、toupper\(\) の使用例」\(p45\)](#) を参照してください。

第 7 章 div_t.h

div_t.h ヘッダファイルは数学計算用の 2 つの構造体を定義しています。

div_t.h の概要

div_t.h ヘッダファイルは、2 つの構造体から構成されます。

[「div_t」\(p47\)](#): div 関数からの余りと商の変数を記憶する

[「ldiv_t」\(p47\)](#): ldiv 関数からの余りと商の変数を記憶する

div_t

div 関数からの余りと商を記憶します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <div_t.h>`

```
typedef struct {
    int quot;
    int rem;
} div_t;
```

参照 [「div」\(p255\)](#)

ldiv_t

div 関数からの余りと商を記憶します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <div_t.h>`

```
typedef struct {
    int quot;
    int rem;
} ldiv_t;
```

参照 [「ldiv」\(p260\)](#)

第 8 章 errno.h

errno.h ヘッドファイルは大域変数エラーコード変数 extern errno を提供します。

errno.h の概要

errno.h には大域的に宣言された変数が含まれています。

[「errno」\(p49\)](#)

errno

errno.h ヘッドファイルは大域エラーコード変数 errno を提供します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <errno.h>`
`extern int errno;`

警告！ PowerPC Mac OS および Windows 用の math ライブラリは（最適化した場合）1990 ANSI C 規格（math 関数が errno に設定されていない）と完全互換ではありません。MSL math ライブラリは優れたエラー検知方法を提供します。fpclassify（完全にポータブルです）によってエラーレポート機能を利用できます。errno の設定は非効率的であるため、旧式の方法と考えられています。68k Mac OS ではさまざまな math 機能に errno を設定できます。

標準ライブラリでのほとんどの関数は、エラーが発生したときに特別な値を返します。プログラマはしばしばエラーの特徴について知る必要があります。いくつかの関数は、大域変数 errno に値を割り当てることによって詳細なエラー情報を提供します。errno 変数は errno.h ヘッドファイルで宣言されています。[「エラー値の定義」\(p50\)](#)を参照してください。

例えば math.h の double pow(double x, double y) 関数は、xy の浮動小数点の値を計算します。式 0ⁿ は、n がゼロ以下であれば double 型を使って表現できません。引数 x が 0.0 で、引数 y が 0.0 以下であれば、pow() 関数は EDOM（Domain error）の値を errno に設定し、0.0 を返します。

errno 変数は、関数呼び出しが成功した時にはクリアされません。値は、errno を使う関数がそれ自身のエラーの値を返す時のみ変更されます。errno を使う関数呼び出しの前に 0 を errno へ割り当てるのはプログラマの責任において行ってください。使用例は[例 8.1 \(p50\)](#) を参照してください。

表 8.1 エラー値の定義

errno の値	説明
EDOM	領域エラー。関数へ渡される引数は、正当な範囲ではない。
ERANGE	範囲エラー。関数は、その型で表現された値を返すことができない。
ENOERR	エラーなしはゼロと等しい。
EPOS	ストリーム上での位置エラー。
ESIGPARM	シグナル引数のエラー。
nonzero value	標準の C 関数によって使用される。
Win32 のみ	説明
EPERM	アクセス権エラー。
EACCES	アクセス権が否定された。
EBADF	不当なファイル番号。
EDEADLOCK	リソースのデッドロックが発生しない。
EMFILE	オープンされているファイルが多すぎる。
ENOENT	ファイルあるいはディレクトリが見つからない。
ENFILE	ファイルがない。
ENOSPC	デバイスに空き領域がない。
EINVAL	無効な引数。
EIO	入出力エラー。
ENOMEM	メモリ不足。
ENOSYS	システムが存在しないエラー。
BeOS のみ	説明
EOK	EOK は、ENOERR と等しい。

例 8.1 errno の例

```
#include <errno.h>
#include <stdio.h>
#include <math.h>

int main(void)
{
```

```
double x, y, result;

printf("Enter two floating point values.\n");
scanf("%lf %lf", &x, &y);
errno = 0; // 命令を実行する前に errno をリセットします。
result = pow(x, y);

if (errno == EDOM)
    printf("Domain error!\n");
else
    printf("%f to the power of %f is %f.\n", x, y, result);

return 0;
}

/* 出力 :
Enter two floating point values.
1.2
3.4
1.200000 to the power of 3.400000 is 1.858730.
*/
```


第 9 章 fcntl.h

ヘッダファイル `fcntl.h` には、UNIX プログラムから移植するための便利なファイル制御関数があります。

fcntl.h の概要

`fcntl.h` ヘッダファイルには、以下の関数があります。

- 「[creat](#)」(p53): ファイルの生成
- 「[fcntl](#)」(p54): ファイル制御記述子
- 「[open](#)」(p56): ファイルのオープン
- 「[umask](#)」(p58): ファイル許可のマスク

fcntl.h と UNIX の互換性

`fcntl.h` ヘッダファイルには、UNIX からの移植に役立つ関数があります。これらの関数は、多くの UNIX ライブラリの関数に類似しています。しかし UNIX と Mac OS オペレーティングシステムは基本的に異なるため、同一ではありません。以下の節では関数の異なる点についても説明します。

一般に新しいプログラムでは、代わりに本来の API の代用プログラムを使うため、これらの関数は使わないほうがよいでしょう。

注意: UNIX あるいは DOS プログラムを移植中であれば、その他の UNIX 互換のヘッダの関数が必要かもしれません。

creat

新しいファイルを生成したり、現存ファイルへ上書きします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <fcntl.h>`
`int creat(const char *filename, int mode);`

引数 この関数の引数です。

filename	int	作成中のファイル名
mode	int	オープンモード

注意点 この関数は書き込み可能な `filename` という名前のファイルを作成します。ファイルが存在しない場合、新たに生成します。ファイルが既に存在する場合、上書きします。関数は引数 `mode` を無視します。

以下の関数呼び出し

```
creat(path, mode);
```

は、以下の関数呼び出し

```
open(path, O_WRONLY|O_CREAT|O_TRUNC, mode);
```

と等しいです。

戻り値 成功した場合、生成されたファイルのファイル記述子を返します。エラーが発生した場合、`-1` を返します。

参照 [「fopen」\(p184\)](#)、[「fdopen」\(p341\)](#)、[「open」\(p56\)](#)、[「close」\(p322\)](#)

例 9.1 creat() の使用例

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    int fd;

    fd = creat("Jeff:Documents:mytest", 0);
    /* ボリューム Jeff のフォルダ Documents 内に mytest
       という名前の新しいファイルを作成します。 */

    write(fd, "Hello world!\n", 13);
    close(fd);
    return 0;
}
```

fcntl

ファイル記述子を実行します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <fcntl.h>`
`int fcntl(int fildes, int cmd, ...);`

引数 この関数の引数です。

fildes	int	ファイルの説明
cmd	int	ファイルシステムへのコマンド
...		変数の引数リスト

注意点 この関数はファイル記述子 `fildes` に対して、`cmd` で指定されたコマンドを実行します。

CodeWarrior の ANSI ライブラリでは、`fcntl()` は、1 つだけのコマンド、`F_DUPFD` を実行できます。このコマンドは、`fildes` が参照するファイルのファイル記述子の複写を返します。関数呼び出しでは 3 番目の引数を持たなければなりません。新しいファイル記述子は 3 番目の引数と同じか、それ以上のところを基底として利用可能なファイル記述子です。

戻り値 成功した場合は `fcntl()` はファイル記述子を返します。エラーの場合は -1 を返します。

参照 [「fileno」\(p342\)](#)、[「open」\(p56\)](#)、[「fdopen」\(p341\)](#)

例 9.2 fcntl() の使用例

```
#include <unix.h>

int main(void)
{
    int fd1, fd2;

    fd1 = open("mytest", O_WRONLY | O_CREAT);

    write(fd1, "Hello world!\n", 13);
    /* 元のファイル記述子へ書き込みます。    */

    fd2 = fcntl(fd1, F_DUPFD, 0);
    /* ファイル記述子の複写を生成。ファイル記述子の複写を生成。
成。    */

    write(fd2, "How are you doing?\n", 19);
    /* 複写されたファイル記述子へ書き込みます。    */

    close(fd2);

    return 0;
}

/* このプログラムを実行した後、ファイル mytest は以下ようになります。
Hello world!
```

```
How are you doing?  
*/
```

open

ファイルをオープンしてその id を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

```
プロトタイプ #include <fcntl.h>  
int open(const char *path, int oflag);
```

引数 この関数の引数です。

path	char *	文字列としてのファイルパス
oflag	int	オープンモード

注意点 関数 open() は、システムレベルの入力と出力に対応したファイルを生成します。このファイルはUNIX ライクな関数 read() と write() を用いて使用されます。

表 9.1 open() を使った規定のファイルオープンモード

モード	説明
O_RDWR	読み書き兼用のファイルのオープン。
O_RDONLY	読み込み用のファイルのオープン。
O_WRONLY	書き込み用のファイルのオープン。
O_APPEND	ファイルの最後に追加するためにファイルをオープン。
O_CREAT	ファイルが存在しない場合はファイルを生成する。
O_EXCL	ファイルが存在する場合はファイルを生成しない。
O_TRUNC	オープンの後にファイルを削除する。
O_NRESOLVE	いかなるエイリアスも解析しない。
O_ALIAS	エイリアスファイルをオープンする（ファイルがエイリアスの場合）。
O_RSRC	リソースフォークをオープン。
O_BINARY	バイナリモードでファイルをオープンする（デフォルトはテキストモード）。
F_DUPFD	複写されたファイル記述子を返す。

戻り値 open() は整数値としてファイル id を返します。

参照 [「close」\(p322\)](#)、[「lseek」\(p331\)](#)、[「read」\(p332\)](#)、[「write」\(p336\)](#)

例 9.3 open() の使用例

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

#define SIZE FILENAME_MAX
#define MAX 1024

char fname[SIZE] = "DonQ.txt";

int main(void)
{
    int fdes;
    char temp[MAX];
    char *Don = "In a certain corner of la Mancha, the name of\n\
which I do not choose to remember,...";
    char *Quixote = "there lived\nnone of those country\
gentlemen, who adorn their\nhalls with rusty lance\
and worm-eaten targets.";

    /* printfのためのヌルで終わる temp 配列 */
    memset(temp, '\\0', MAX);

    /* ファイルのオープン */
    if((fdes = open(fname, O_RDWR | O_CREAT ))== -1)
    {
        perror("Error ");
        printf("Can not open %s", fname);
        exit( EXIT_FAILURE);
    }

    /* ファイルへ書き込む */
    if( write(fdes, Don, strlen(Don)) == -1)
    {
        printf("%s Write Error\n", fname);
        exit( EXIT_FAILURE );
    }

    /*move back to over write ... characters */
    if( lseek( fdes, -3L, SEEK_CUR ) == -1L)
    {
        printf("Seek Error");
    }
}
```

```
        exit( EXIT_FAILURE );
    }

    /* write to a file */
    if( write(fdes, Quixote, strlen(Quixote)) == -1)
    {
        printf("Write Error");
        exit( EXIT_FAILURE );
    }

    /* 読み込みのため、ファイルの最初へ移動 */
    if( lseek( fdes, 0L, SEEK_SET ) == -1L)
    {
        printf("Seek Error");
        exit( EXIT_FAILURE );
    }

    /* ファイルを読み込む */
    if( read( fdes, temp, MAX ) == 0)
    {
        printf("Read Error");
        exit( EXIT_FAILURE );
    }

    /* ファイルをクローズ */
    if(close(fdes))
    {
        printf("File Closing Error");
        exit( EXIT_FAILURE );
    }

    puts(temp);

    return 0;
}
```

In a certain corner of la Mancha, the name of which I do not choose to remember, there lived one of those country gentlemen, who adorn their halls with rusty lance and worm-eaten targets.

umask

UNIX 形式のファイル生成マスクを設定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <fcntl.h>`
`mode_t umask(mode_t cmask);`

引数 この関数の引数です。

`cmask` `mode_t` ビットマスクの許可

注意点 関数 `umask` は `open()`、`creat()`、`mkdir()` を呼び出すために使われます。`mode` argument のビット許可をオフにします。

注意： ビット許可は Mac OS でも Windows でも使用されません。この関数はコンパイルと互換性のために提供されています。

戻り値 以前のマスク。Mac OS および Windows オペレーティングシステムではゼロを返します。

参照 [「creat」\(p53\)](#)、[「open」\(p56\)](#)、[「mkdir」\(p160\)](#)

第 10 章 float.h

float.h ヘッダファイルマクロは、float、double、long double 型に関する「[浮動小数点数の指数部](#)」(p61) を指定します。

float.h の概要

浮動小数点数の指数部

float.h ヘッダファイルは、float、double、long double 型に関して表現された浮動小数点数の指数部を指定するマクロから構成されています。

これらのマクロは「[浮動小数点の指数部](#)」(p61) に列挙されています。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

「[浮動小数点の指数部](#)」(p61) では、float.h で定義されたマクロを列挙しています。FLT で始まるマクロは float 型で使い、DBL では double 型、そして LDBL では long double 型で使います。

FLT_RADIX マクロは指数表現の基数を指定します。

FLT_ROUNDS は丸めモードを指定します。CodeWarrior C では、正の無限大に向かって丸めています。

表 10.1 浮動小数点の指数部

マクロ	説明
FLT_MANT_DIG, DBL_MANT_DIG, LDBL_MANT_DIG	基数 FLT_RADIX の有効数字
FLT_DIG, DBL_DIG, LDBL_DIG	10 進数の精度
FLT_MIN_EXP, DBL_MIN_EXP, LDBL_MIN_EXP	FLT_RADIX で生成、表現可能なもっとも小さい負の整数の指数

マクロ	説明
FLT_MIN_10_EXP, DBL_MIN_10_EXP, LDBL_MIN_10_EXP	10 を基数として生成、表現可能なもっとも小さい負の整数の指数
FLT_MAX_EXP, DBL_MAX_EXP, LDBL_MAX_EXP	FLT_RADIX で生成、表現可能なもっとも大きい正の整数の指数
FLT_MAX_10_EXP, DBL_MAX_10_EXP, LDBL_MAX_10_EXP	10 を基数として生成、表現可能な最も大きい正の整数の指数
FLT_MIN, DBL_MIN, LDBL_MIN	最も小さい正の浮動小数点の値
FLT_MAX, DBL_MAX, LDBL_MAX	最も大きい正の浮動小数点の値
FLT_EPSILON, DBL_EPSILON, LDBL_EPSILON	最も小さい表現可能な小数部

第 11 章 FSp_fopen.h

FSp_fopen.h ヘッドファイルは FSp_fopen 関数を宣言しています。

FSp_fopen.h の概要

FSp_fopen.h ヘッドファイルには以下の関数が含まれています。

[「FSp_fopen」\(p63\)](#): open に関する Mac OS のファイルオープン

FSp_fopen

FSpec を使ったファイルのオープンと FILE ポインタを返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <Fsp_fopen.h>
FILE * FSp_fopen (ConstFSSpecPtr spec,
                  const char * open_mode);
```

引数 この関数の引数です。

spec	ConstFSSpecPtr	Toolbox ファイルポインタ
open_mode	char *	オープンモード

説明と注意点 関数 FSp_fopen は、Macintosh Toolbox の FSpec 関数を使ってファイルをオープンし、FILE ポインタを返します。

注意： この関数を使うには、関連する FSp_fopen.c ソースファイルをプロジェクトに追加する必要があります。これは MSL C ライブラリには含まれていません。

戻り値 FSp_fopen 機能は、FILE ポインタを返します。

参照 [「fopen」\(p184\)](#)

第 12 章 io.h

io.h ヘッダでは、Windows のコンソールの関数を定義します。

io.h の概要

io.h ヘッダファイルは以下のものから構成されています。

- 「[_chdir](#)」(p65): ディレクトリを変更する
- 「[_chdrive](#)」(p66): ドライブを変更する
- 「[_fileno](#)」(p66): ファイルハンドルを返す
- 「[_getcwd](#)」(p67): 現在の作業ディレクトリを取得する
- 「[_GetHandle](#)」(p67): デバイスハンドルを取得する
- 「[_get_osfhandle](#)」(p66): オペレーティングシステムのファイルハンドルを取得する
- 「[_heapmin](#)」(p68): 使用していないヒープ領域をシステムへ解放する
- 「[_isatty](#)」(p68): デバイスが文字型デバイスかどうかを判定する
- 「[_makepath](#)」(p69): パスを生成する
- 「[_open_osfhandle](#)」(p69): OS のファイルハンドラをオープンする
- 「[_searchenv](#)」(p70): ファイルに関する環境を検索する

_chdir

この関数は、ディレクトリの変更する場合に使います。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <io.h>
int _chdir(const char *dirname);
```

引数 この関数の引数を以下に示します。

dirname const char * 新しいディレクトリ名

戻り値 成功した場合は偽を返し、失敗した場合は `errno` 変数を設定して -1 を返します。

参照 [「_chdrive」\(p66\)](#)、[「_makepath」\(p69\)](#)

_chdrive

この関数は、ドライブを変更する場合に利用します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <io.h>
int _chdrive(int drive);
```

引数 この関数の引数を以下に示します。

drive int 変更先のドライブ

戻り値 成功した場合は Chdrive は偽を返し、失敗した場合は真を返します。

参照 [「_chdir」\(p65\)](#)、[「_makepath」\(p69\)](#)

_fileno

この関数は、ファイルハンドルの ID を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <io.h>
int _fileno(FILE *stream);
```

引数 この関数の引数を以下に示します。

stream FILE ID を見つけるストリーム

説明と注意点 引数 stream がファイルを指定していない、またはファイルをオープンしていない場合、結果は定まりません。

戻り値 ファイルハンドルの ID を返します。

参照 [「_get_osfhandle」\(p66\)](#)

_get_osfhandle

この関数はオペレーティングシステムのファイルハンドルを取得します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <io.h>
long _get_osfhandle(int filehandle);
```

引数 この関数の引数を以下に示します。

filehandle	int	ファイルハンドル
------------	-----	----------

戻り値 成功した場合はオペレーティングシステムのファイルハンドルを返し、失敗した場合は `errno` を設定して NULL を返します。

参照 [「_fileno」\(p66\)](#)、[「_open_osfhandle」\(p69\)](#)

_getcwd

現在の作業ディレクトリを取得します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <io.h>
char * _getcwd(
    char *path,
    int maxlen);
```

引数 この関数の引数を以下に示します。

path	char *	文字列を記憶するバッファ
maxlen	int	バッファの最大の長さ

戻り値 成功した場合、バッファ `path` へのポインタを返します。失敗した場合、`errno` 変数を設定して NULL を返します。

参照 [「_searchenv」\(p70\)](#)

GetHandle

`GetHandle` は現在のオブジェクトハンドラを検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <io.h>`
`int GetHandle();`

引数 なし。

戻り値 デバイスハンドラを返します。

参照 「[_isatty](#)」(p68)

`_heapmin`

この関数はヒープメモリをシステムに解放します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <io.h>`
`int _heapmin(void);`

引数 なし。

戻り値 成功した場合、`heapmin` はゼロを返します。それ以外は、`errno` を `ENOSYS` に設定して、`-1` を返します。

`_isatty`

この関数はデバイスが文字型デバイスかどうかを判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `int _isatty(int fileno);`

引数 この関数の引数を以下に示します。

<code>fileno</code>	<code>int</code>	デバイス ID
---------------------	------------------	---------

戻り値 デバイスが文字型デバイスの場合は真を返し、それ以外は偽を返します。

参照 「[GetHandle](#)」(p67)

_makepath

Makepath はパスを生成する場合に使用します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <io.h>
void _makepath(
    char *path,
    const char *drive,
    const char *dir,
    const char *fname,
    const char *ext);
```

引数 この関数の引数を以下に示します。

path	char *
drive	const char *
dir	const char *
fname	const char *
ext	const char *

戻り値 なし。

参照 [「_chdir」\(p65\)](#)、[「_chdrive」\(p66\)](#)

_open_osfhandle

open_osfhandle は、オペレーティングシステムのファイルハンドルをオープンします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <io.h>
int _open_osfhandle(long ofshandle, int flags);
```

引数 この関数の引数を以下に示します。

ofshandle	long	オープンするハンドル
flags	int	モード

戻り値 成功した場合はハンドルを返し、それ以外は -1 を返します。

参照 [「GetHandle」\(p67 \)](#)、[「_get_osfhandle」\(p66 \)](#)

_searchenv

`searchenv` は環境変数で定義されるパスを使ってファイルを検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <io.h>
void _searchenv(
    const char *filename,
    const char *varname,
    char *pathname);
```

引数 この関数の引数を以下に示します。

<code>filename</code>	<code>const char *</code>	検索するファイル名
<code>varname</code>	<code>const char *</code>	環境変数
<code>pathname</code>	<code>char *</code>	ファイル名

戻り値 なし。

参照 [「_getcwd」\(p67 \)](#)

第 13 章 limits.h



limits.h ヘッドファイルのマクロは、整数型の最大と最小の制限を定義しています。

limits.h の概要

limits.h ヘッドファイルは以下で列挙されているマクロから構成されます。

[「整数型の限界」\(p71\)](#)

整数型の限界

limits.h ヘッドファイルのマクロでは、整数型の最大と最小の制限を定義しています。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

[「整数の制限」\(p71\)](#) はこのマクロについて説明します。

表 13.1 整数の制限

マクロ	説明
CHAR_BIT	bit フィールドではない最小ビット数
CHAR_MAX	char 型に関する最大値
CHAR_MIN	char 型に関する最小値
SCHAR_MAX	signed char 型に関する最大値
SCHAR_MIN	signed char 型に関する最小値
UCHAR_MAX	unsigned char 型に関する最大値
SHRT_MAX	short int 型の最大値
SHRT_MIN	short int 型の最小値
USHRT_MAX	unsigned short int 型の最大値
INT_MAX	int 型の最大値
INT_MIN	int 型の最小値
LONG_MAX	long int 型の最大値

マクロ	説明
LONG_MIN	long int 型の最小値
ULONG_MAX	unsigned long int 型の最大値

第 14 章 locale.h

locale.h ヘッドファイルは異なった文字セット、数値、通貨書式を扱うための機能を提供します。

locale.h の概要

[「地域設定情報の仕様」\(p73\)](#) の操作に関して使用可能な機能を以下に示します。

[「lconv 構造体と localeconv\(\) の戻り値」\(p73\)](#)

[「localeconv」\(p74\)](#): 地域設定情報を取得する

[「setlocale」\(p74\)](#): 地域設定情報を設定する

地域設定情報の仕様

ANSI C の標準規格では、C コンパイラのある部分が地域の差に適応可能なように規定されています。locale.h ヘッドファイルは文字セット、数値、通貨フォーマットの違いを扱うための機能を提供しています。CodeWarrior C は、C の地域設定情報のみをサポートしています。

locale.h で定義される lconv 構造体は、数値を文字型に変換するために、数値、通貨書式の特徴を指定しています。localeconv() に対する呼び出しでは、C の地域情報 ([例 14.1](#)) に関する設定を持つ lconv 構造体のポインタを返します。現在の地域設定情報に適用できない場合、lconv のメンバに [「CHAR_MAX」\(p71\)](#) の値を割り当てます。

例 14.1 lconv 構造体と localeconv() の戻り値

```
struct lconv {
    char *currency_symbol;
    char *int_curr_symbol;
    char *mon_decimal_point;
    char *mon_grouping;
    char *mon_thousands_sep;
    char *negative_sign;
    char *positive_sign;
    char frac_digits;
    char int_frac_digits;
    char n_cs_precedes;
    char n_sep_by_space;
    char n_sign_posn;
    char p_cs_precedes;
    char p_sep_by_space;
    char p_sign_posn;
```

```
char *decimal_point;  
char *grouping;  
char *thousands_sep;  
};
```

localeconv

現在の地域情報に関する lconv の設定を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <locale.h>  
struct lconv *localeconv(void);
```

引数 なし。

戻り値 localeconv() は、C の地域設定情報に関する lconv 構造体へのポインタを返します。

setlocale

C コンパイラに対する地域設定情報の問い合わせと設定を行います。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <locale.h>  
char *setlocale(  
    int category,  
    const char *locale);
```

引数 この関数の引数です。

category	int	問い合わせや設定のための、C コンパイラのパート
locale	char *	locale へのポインタ

説明と注意点 引数 category は、問い合わせや設定のための C コンパイラの特定のパートを指定します。

引数には、locale.h でマクロとして定義されている 6 個の値、LC_ALL (全てのパート)、LC_COLLATE (照合関数 strcoll())、LC_CTYPE (ctype.h の関数と stdlib.h のマルチバイトの変換関数)、LC_MONETARY (通貨書式)、LC_NUMERIC (数値書式)、LC_TIME (時間と日付の書式) のいずれか 1 つを持つことができます。

引数 `locale` がヌルポインタ、または空の文字列である場合、問い合わせモードです。
`setlocale()` 関数は、どの地域情報がコンパイラの特定のパートへ設定されるかを指す文字配列へのポインタを返します。CodeWarrior C コンパイラは、C の地域設定情報をサポートしています。

CodeWarrior C コンパイラの地域設定情報のパート設定の試みは、効果がありません。

参照 [「strcoll」\(p284\)](#)

第 15 章 malloc.h

このヘッダは 1 個の関数 [alloca](#) を宣言します。この関数はスタックから素早くメモリを割り当てます。

malloc.h の概要

malloc.h ヘッダファイルは次のものから構成されます。

[「alloca」\(p77\)](#)：スタックからメモリを割り当てる

alloca

スタック上のメモリを素早く割り当てます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <malloc.h>
void *alloca(size_t nbytes);
```

引数 この関数の引数です。

nbytes size_t 割り当てバイトのサイズ

注意点 この関数は nbytes 長のメモリブロックを指すポインタを返します。ブロックは関数のスタック上に割り当てられます。この関数は、現在のスタックポインタを減少させるので素早く動作します。関数が終了したとき、主記憶は自動的に解放されます。

alloca() を多くの主記憶の割り当てのために使用する場合、プロジェクト設定パネルでプロジェクトのスタックサイズを増加させてください。

戻り値 成功した場合、alloca() はメモリブロックのポインタを返します。エラーが発生した場合、alloca() は NULL を返します。

参照 [「calloc」\(p253\)](#)、[「free」\(p257\)](#)、[「malloc」\(p260\)](#)、[「realloc」\(p265\)](#)

第 16 章 math.h

math.h ヘッドファイルは浮動小数点の数学的な変換関数を提供します。

math.h の概要

math.h ヘッドファイルでは次の関数が宣言されています。

マクロ

[「fpclassify」\(p82\)](#) : 浮動小数点の数値を分類する

[「isfinite」\(p83\)](#) : 有限な数値をテストする

[「isnan」\(p83\)](#) : 計算可能な数値をテストする

[「isnormal」\(p84\)](#) : 通常の数値をテストする

[「signbit」\(p84\)](#) : 負の数値をテストする

関数

[「acos」\(p85\)](#) : 逆余弦 (arccosine) を決定する

[「asin」\(p86\)](#) : 逆正弦 (arcsine) を決定する

[「atan」\(p86\)](#) : 逆正接 (arctangent) を決定する

[「atan2」\(p87\)](#) : 2 変数の逆正接 (arctangent) を決定する

[「ceil」\(p88\)](#) : x 以下の最小の整数 (切り捨て) を決定する

[「cos」\(p89\)](#) : 余弦 (cosine) を決定する

[「cosh」\(p90\)](#) : 双曲線余弦 (hyperbolic cosine) を決定する

[「exp」\(p91\)](#) : 指数関数を計算する

[「fabs」\(p92\)](#) : 絶対値を決定する

[「floor」\(p93\)](#) : x 以上の最大の int 型 (切り上げ) を決定する

[「fmod」\(p94\)](#) : 除算の剰余を決定する

[「frexp 実装」\(p95\)](#) : 仮数と指数の値を抽出する

[「ldexp」\(p99\)](#) : 仮数と指数から値を計算する

[「log」\(p100\)](#) : 自然対数を決定する

[「log10」\(p101\)](#) : 10 を底とする対数を決定する

[「modf」\(p102\)](#) : 整数部と少数部に分ける

[「pow」\(p103\)](#): ベキ乗について言及する

[「sin」\(p104\)](#): 正弦 (sine) を決定する

[「sinh」\(p105\)](#): 双曲線正弦 (hyperbolic sine) を決定する

[「sqrt」\(p106\)](#): 二乗根を決定する

[「tan」\(p108\)](#): 正接 (tangent) を決定する

[「tanh」\(p109\)](#): 双曲線正接 (hyperbolic tangent) を決定する

C9X の実装

[「acosh」\(p110\)](#): 逆双曲線余弦 (arc hyperbolic cosine、負の値ではない) を計算する

[「asinh」\(p111\)](#): 逆双曲線正弦 (hyperbolic sine) を求める

[「atanh」\(p111\)](#): 逆双曲線正接 (arc hyperbolic tangent) を求める

[「copysign」\(p111\)](#): x の大きさと y の符号を持つ値を求める

[「erf」\(p112\)](#): 誤差関数を求める

[「erfc」\(p112\)](#): 補充誤差関数を求める

[「exp2」\(p113\)](#): 2 を基底とした指数関数を求める

[「expm1」\(p113\)](#): -1 の指数関数を求める

[「fdim」\(p114\)](#): その引数の正の差分を求める

[「fmax」\(p114\)](#): 引数の値の最大の数値を求める

[「fmin」\(p114\)](#): 引数の値の最小の数値を求める

[「gamma」\(p115\)](#): ガンマ関数を求める

[「hypot」\(p115\)](#): 引数の 2 乗の和の平方根を求める

[「isgreater」\(p97\)](#): x が y よりも大きい 2 つの数値を比較する

[「isgreaterless」\(p97\)](#): x が y と等しくない 2 つの数値を比較する

[「isless」\(p98\)](#): x が y よりも小さい 2 つの数値を比較する

[「islessequal」\(p98\)](#): x が y よりも小さいまたは等しいか、2 つの数値を比較する

[「isunordered」\(p98\)](#): 引数の順番を比較する

[「lgamma」\(p116\)](#): 絶対値の常用対数 (log) を求める

[「log1p」\(p116\)](#): 基底 e の対数を求める

[「log2」\(p117\)](#): 基底 2 の対数を求める

[「logb」\(p117\)](#): double_t の型の指数を抽出する

[「nan」\(p118\)](#): NaN のテストをする

[「nearbyint」\(p118\)](#): 引数が整数になるように四捨五入する

[「nextafter」\(p119\)](#): 関数の型において表現可能な値を決定する

[「remainder」\(p119\)](#): IEC 559 で要求されている、剰余 $x \text{ REM } y$ を求める

[「remquo」\(p120\)](#): 剰余関数と同様の剰余を求める

[「rint」\(p120\)](#): 引数を整数値へ四捨五入する

[「rinttol」\(p121\)](#): 引数を long の整数値に四捨五入する

[「round」\(p121\)](#): 浮動小数点の型で引数を整数値に丸める

[「roundtol」\(p122\)](#): 引数をもっとも近い整数値に丸める

[「scalb」\(p122\)](#): $x * \text{FLT_RADIX}^n$ を求める

[「trunc」\(p123\)](#): 浮動小数点の型で引数を引数以下の整数値に丸める

浮動小数点の計算

math.h で定義される HUGE_VAL マクロが strtod() 関数のエラーの値として返されます。strtod() の情報に関しては [「strtol」\(p268\)](#) を参照してください。

最適化していない x86 math.h 関数は、エラーの状態を示す大域変数 [「errno」](#) を使います。その多くの関数は引数が正当な範囲を越えた場合、errno に EDOM ([表 8.1 \(p50\)](#) を参照) を設定します。

NaN (Not a Number)

NaN とは「Not a Number」の略です。その他の数値とは関係のない数値を意味します。NaN はある数値よりも大きい、小さい、または等しくはありません。無限大と数値の比較は可能です。つまり無限大はすべての数値より大きく、負の無限大はすべての数値よりも小さいです。

signalling NaN と quiet NaN の 2 種類の NaN があります。両者は指数と、少なくとも 1 つのゼロ以外の有効ビットを持ちます。しかし、signalling NaN は 1 として 2 つの最上位ビットを持ち、quiet NaN は 1 として第二の最上位ビットを持ちます。

Quiet NaN

quiet NaN はゼロ ÷ ゼロや、無限大 - 無限大などの不確定演算の結果です。IEEE 浮動小数点規格は、基本算術 (+、/、-、*) または非算術操作 (ロード、ストア) のオペランドとして NaN が出現したときに無効例外を生じることにより、quiet NaN を検知可能であると規定しています。MSL (Metrowerks Standard Library) は IEEE の仕様に準拠します。

Signaling NaN

signalling NaN は算術の結果として起こるものではありません。signalling NaN は不当なメモリ値を、signalling NaN と同じビットパターンを持つ浮動小数点レジスタへロードしたときに生じます。IEEE 754 では、このような場合は無効な例外を発して signalling NaN は quiet NaN へ変換し、signalling NaN の生存期間を短くすることを要求しています。

浮動小数点エラーテスト

PowerPC Mac OS および Windows 用の math ライブラリは (最適化した場合) 1990 ANSI C 規格 (math 関数が errno に設定されていない) と完全互換ではなくなります。

errno の設定は非効率的であるため、旧式の方法と考えられています。68k Mac OS ではさまざまな math 機能に errno を設定できます。

MSL math ライブラリは優れたエラー検知方法を提供します。fpclassify (完全にポータブルです) によってエラーレポート機能を利用できます。[「エラー検出の使用例」\(p83\)](#) にエラーを検知するサンプルコードを示します。これにより fpclassify が返した値に基づいてアルゴリズムを回復することができます。

インラインを積極的に利用する

CodeWarrior の Win32 x86 コンパイラは、[インラインを積極的に利用する (inline intrinsics)] という最適オプションを提供します。このオプションがオンの場合、math 関数は大域変数 errno を設定しません。CodeWarrior がビルドした ANSI ライブラリのデバッグバージョンでは、このオプションはオフなので errno が設定されます。最適化したリリースバージョンのライブラリでは、オプションはオンで errno は設定されません。

浮動小数点分類マクロ

浮動小数点エラーの分類を行う機能があります。

列挙定数

MSL には浮動小数点評価用に以下の定数型があります。

- FP_NAN : quiet NaN
- FP_INFINITE : 正または負の無限大
- FP_ZERO : 正または負のゼロ
- FP_NORMAL : すべての正規数値
- FP_SUBNORMAL : 正規以外の数値

参照 [「NaN \(Not a Number\)」\(p81\)](#)

fpclassify

浮動小数点の数値を分類します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
int __fpclassify(long double x);
int __fpclassifyd(double x);
int __fpclassifyf(float x);
```

引数 この関数の引数を以下に示します。

x float、double、long double のいずれか 評価された数値

戻り値 整数値、FP_NAN、FP_INFINIT、FP_ZERO、FP_NORMAL、FP_SUBNORMAL を返します。

参照 [「isfinite」\(p83\)](#)、[「isnan」\(p83\)](#)、[「isnormal」\(p84\)](#)、[「signbit」\(p84\)](#)、[「NaN \(Not a Number\)」\(p81\)](#)

例 16.1 エラー検出の使用例

```
switch(fpclassify(pow(x,y))
{
case FP_NAN: // we know y is not an int and <0
case FP_INFINITY: // we know y is an int <0
case FP_NORMAL: // given x=0 we know y=0
case FP_ZERO:// given x<0 we know y >0
}
```

isfinite

関数 isfinite は有限な数値をテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`
`int isfinite(double x);`

引数 この関数の引数を以下に示します。

x float、double、long double のいずれか 評価された数値

戻り値 定数とした値が有限ならば真、それ以外は偽を返します。

参照 [「fpclassify」\(p82\)](#)

isnan

isnan 関数は、計算可能な数値をテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
int isnan (double x);
```

引数 この関数の引数を以下に示します。

x float、double、long double のいずれか 評価された数値

戻り値 引数が数値でなければ真を返します。

参照 [「fpclassify」\(p82 \)](#)、[「NaN \(Not a Number \)」\(p81 \)](#)

isnormal

通常の数値をテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
int isnormal(double x);
```

引数 この関数の引数を以下に示します。

x float、double、long double のいずれか 評価された数値

戻り値 引数が通常の数値であれば真を返します。

参照 [「fpclassify」\(p82 \)](#)

signbit

符号付きビットを含む数値をテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>

int __signbit(long double x);
int __signbitd(double x);
```

```
int __signbit(float x);
```

引数 この関数の引数を以下に示します。

x float、double、long double のいずれか 評価された数値

戻り値 引数の値の符号が負であれば真を返します。

参照 [「fpclassify」\(p82\)](#)

浮動小数点 Math 関数

acos

逆余弦 (arccosine) 関数です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
double acos(double x);  
float acosf(float);  
long double acosl(long double);
```

引数 この関数の引数を以下に示します。

x double、float、long double のいずれか 計算用の値

注意点 この関数は逆余弦の値を計算します。

関数 `acos()` は、引数が -1 から +1 の間でない場合、`errno` を `EDOM` に設定します。使用例は [「acos\(\)、asin\(\)、atan\(\)、atan2\(\) の使用例」\(p88\)](#) を参照してください。

戻り値 `acos()` は、ラジアンとしての引数 `x` の逆余弦を返します。`acos()` に対する引数の範囲が -1 から +1 の間の場合、大域変数 `errno` を `EDOM` に設定し、0 を返します。

参照 [「インラインを積極的に利用する」\(p82\)](#)、[「cos」\(p89\)](#)、[「errno」\(p49\)](#)

acosf

`float` 型の値のための `acos()` 関数を実装します。[「acos」\(p85\)](#) を参照してください。

acosl

`long double` 型の値のための `acos()` を実装します。[「acos」\(p85\)](#) を参照してください。

asin

逆正弦 (arcsine) 関数です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `double asin(double x);`
`float asinf(float);`
`long double asinl(long double);`

引数 この関数の引数を以下に示します。

x double、float、long double のいずれか 計算用の値

注意点 この関数は、逆正弦の値を計算します。

関数 `asin()` は、引数が -1 から +1 の間でない場合、`errno` を EDOM に設定します。使用例は「[acos\(\)、asin\(\)、atan\(\)、atan2\(\) の使用例](#)」(p88) を参照してください。

戻り値 関数 `asin()` は、ラジアンとしての x の逆正弦を返します。`asin()` への引数が -1 から +1 の範囲でない場合、大域変数 `errno` を EDOM に設定し、0 を返します。

参照 「[インラインを積極的に利用する](#)」(p82)、[「sin」](#)(p104)、[「errno」](#)(p49)

asinf

`float` 型の値に関する `asin()` 関数を実装します。「[asin](#)」(p86) を参照してください。

asinl

`long double` 型の値に関する `asin()` 関数を実装します。「[asin](#)」(p86) を参照してください。

atan

逆正接 (arctangent) 関数です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

`double atan(double x);`
`float atanf(float);`

```
long double atanl(long double);
```

引数 この関数の引数を以下に示します。

x	float、double、long double のいずれか	計算用の値
---	--------------------------------	-------

注意点 この関数は引数の逆正接の値を計算します。使用例に関しては「[acos\(\)、asin\(\)、atan\(\)、atan2\(\)の使用例](#)」(p88)を参照してください。

戻り値 関数 `atan()` は、ラジアン $[-\pi/2, +\pi/2]$ の範囲の引数 `x` の逆正接を返します。

$[-\pi/2, +\pi/2]$ radians.

参照「tan」(p108)、「errno」(p49)

atanf

float 型の値に関する atan() を実装します。「atan」(p86)を参照してください。

atanl

long double 型の値に関する atan() を実装します。「atan」(p86) を参照してください。

atan2

逆正接（arctangent）関数です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

```
プロトタイプ #include <math.h>
```

```
double atan2(double y, double x);
float atan2f(float, float);
long double atan2l(long double, long double);
```

引数 この関数の引数を以下に示します。

y	double、float、long double のいずれか	値 1
x	double、float、long double のいずれか	値 2

注意点 この関数は両方の引数を使って、 x/y の正接の値を計算します。使用例は「[r.acos\(\)、r.asin\(\)、r.atan\(\)、r.atan2\(\) の使用例](#)」(p88)を参照してください。

x と y の両方がゼロの場合、領域エラーが発生します。

戻り値 関数 `atan2()` は、ラジアン $[-\pi, +\pi]$ の範囲の y/x の逆正接を返します。

$[-\pi, ++ \text{radians}]$.

参照 [「インラインを積極的に利用する」\(p82\)](#)、[「tan」\(p108\)](#)、[「errno」\(p49\)](#)

例 16.2 acos()、asin()、atan()、atan2() の使用例

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double x = 0.5, y = -1.0;

    printf("arccos (%f) = %f\n", x, acos(x));
    printf("arcsin (%f) = %f\n", x, asin(x));
    printf("arctan (%f) = %f\n", x, atan(x));
    printf("arctan (%f / %f) = %f\n", y, x, atan2(y, x));

    return 0;
}
```

Output:

```
arccos (0.500000) = 1.047198
arcsin (0.500000) = 0.523599
arctan (0.500000) = 0.463648
arctan (-1.000000 / 0.500000) = -1.107149
```

atan2f

float 型の値に関する atan2() を実装します。[「atan2」\(p87\)](#) を参照してください。

atan2l

long double 型の値に関する atan2() を実装します。[「atan2」\(p87\)](#) を参照してください。

ceil

x 以上の最小の整数 (切り上げ) を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double ceil(double x);
float ceilf(float);
```



```
long double ceill(long double);
```

引数 この関数の引数を以下に示します。

x

float、double、long double のいずれか

計算用の値

戻り値 `ceil()` は `x` 以上の最小の整数を返します。

参照 [「floor」\(p93\)](#)、[「fmod」\(p94\)](#)、[「round」\(p121\)](#)

例 16.3 `ceil()` の使用例

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double x = 100.001, y = 9.99;

    printf("The ceiling of %f is %f.\n", x, ceil(x));
    printf("The ceiling of %f is %f.\n", y, ceil(y));

    return 0;
}
```

Output:
The ceiling of 100.001000 is 101.000000.
The ceiling of 9.990000 is 10.000000.

`ceilf`

`float` 型の値に関する `ceil()` 関数を実装します。[「ceil」\(p88\)](#) を参照してください。

`ceil`

`long double` 型の値に関する `ceil()` 関数を実装します。[「ceil」\(p88\)](#) を参照してください。

`cos`

余弦 (cosine) を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double cos(double x);  
float cosf(float);  
long double cosl(long double);
```

引数 この関数の引数を以下に示します。

x float、double、long double のいずれか 計算用の値

戻り値 `cos()` は、`x` の余弦を返します。`x` はラジアンとして評価されます。

参照 [「sin」\(p104\)](#)、[「tan」\(p108\)](#)

例 16.4 `cos()` の使用例

```
#include <math.h>  
#include <stdio.h>  
  
int main(void)  
{  
    double x = 0.0;  
    printf("The cosine of %f is %f.\n", x, cos(x));  
  
    return 0;  
}
```

Output:
The cosine of 0.000000 is 1.000000.

`cosf`

`float` 型の値に関する `cos()` 関数を実装します。[「cos」\(p89\)](#) を参照してください。

`cosl`

`long double` 型の値に関する `cos()` 関数を実装します。[「cos」\(p89\)](#) を参照してください。

`cosh`

双曲線余弦 (hyperbolic cosine) を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
double cosh(double x);  
float coshf(float);  
long double coshl(long double);
```

引数 この関数の引数を以下に示します。

x float、double、long double のいずれか 計算される値

戻り値 `cosh()` は引数 `x` の双曲線余弦を返します。

参照 [「インラインを積極的に利用する」\(p82\)](#)、[「sinh」\(p105\)](#)、[「tanh」\(p109\)](#)

例 16.5 `cosh()` の使用例

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double x = 0.0;

    printf("Hyperbolic cosine of %f is %f.\n",x,cosh(x));

    return 0;
}
```

Output:
Hyperbolic cosine of 0.000000 is 1.000000.

`coshf`

`float` 型の値に関する `cosh()` 関数を実装します。[「cosh」\(p90\)](#) を参照してください。

`coshl`

`long double` 型の値に関する `cosh()` 関数を実装します。[「cosh」\(p90\)](#) を参照してください。

`exp`

`ex` を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double exp(double x);
float expf(float);
long double expl(long double);
```

引数 この関数の引数を以下に示します。

x float、double、long double のいずれか 計算される値

戻り値 `exp()` は `ex` を返します。e は natural logarithm の基底値です。

注意点 より大きな数値に関して範囲エラーが発生する可能性があります。

参照 [「インラインを積極的に利用する」\(p82\)](#)、[「log」\(p100\)](#)、[「expm1」\(p113\)](#)、[「exp2」\(p113\)](#)、[「pow」\(p103\)](#)

例 16.6 `exp()` の使用例

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double x = 4.0;
    printf("The natural logarithm base e raised to the\n");
    printf("power of %f is %f.\n", x, exp(x));

    return 0;
}
```

Output:
The natural logarithm base e raised to the
power of 4.000000 is 54.598150.

`expf`

float 型の値に関する `exp()` 関数を実装します。[「exp」\(p91\)](#) を参照してください。

`expl`

long double 型の値に関する `exp()` 関数を実装します。[「exp」\(p91\)](#) を参照してください。

`fabs`

浮動小数点の絶対値を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double fabs(double x);
float fabsf(float);
long double fabsl(long double);
```

引数 この関数の引数を以下に示します。

x float、double、long double のいずれか 計算される値

戻り値 fabs() は x の絶対値を返します。

参照 [「floor」\(p93\)](#)、[「ceil」\(p88\)](#)、[「fmod」\(p94\)](#)

例 16.7 fabs() の使用例

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double s = -5.0, t = 5.0;
    printf("Absolute value of %f is %f.\n", s, fabs(s));
    printf("Absolute value of %f is %f.\n", t, fabs(t));

    return 0;
}
```

Output:
Absolute value of -5.000000 is 5.000000.
Absolute value of 5.000000 is 5.000000.

fabsf

float 型の値に関する fabs() 関数を実装します。[「fabs」\(p92\)](#) を参照してください。

fabsl

long double 型の値に関する fabs() 関数を実装します。[「fabs」\(p92\)](#) を参照してください。

floor

x 以上の最大の浮動小数点を求めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double floor(double x);  
float floorf(float);  
long double floorl(long double);
```

引数 この関数の引数を以下に示します。

x float、double、long double のいずれか 計算される値

戻り値 `floor()` は、x 以下の最大の整数（切り捨て）を返します。

参照 [「ceil」\(p88\)](#)、[「fmod」\(p94\)](#)、[「fabs」\(p92\)](#)

例 16.8 floor() の使用例

```
#include <math.h>  
#include <stdio.h>  
  
int main(void)  
{  
    double x = 12.03, y = 10.999;  
  
    printf("Floor value of %f is %f.\n", x, floor(x));  
    printf("Floor value of %f is %f.\n", y, floor(y));  
  
    return 0;  
}
```

Output:
Floor value of 12.030000 is 12.000000.
Floor value of 10.999000 is 10.000000.

floorf

float 型の値に関する `floor()` 関数を実装します。[「floor」\(p93\)](#) を参照してください。

floorl

long double 型の値に関する `floor()` 関数を実装します。[「floor」\(p93\)](#) を参照してください。

fmod

x/y の浮動小数点型の剰余を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double fmod(double x, double y);
float fmodf(float, float);
long double fmodl(long double, long double);
```

引数 この関数の引数を以下に示します。

x	double、float、long double のいずれか	計算用の値
y	double、float、long double のいずれか	除数

戻り値 `fmod()` は、可能な場合、ある整数 `i` に対して $|f| < |y|$ となるような、 $x = i y + f$ の値 `f` を返します。 `f` の符号は `x` の符号に対応します。

参照 [「floor」\(p93\)](#)、[「ceil」\(p88\)](#)、[「fmod」\(p94\)](#)、[「fabs」\(p92\)](#)

例 16.9 `fmod()` の使用例

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double x = -54.4, y = 10.0;
    printf("Remainder of %f / %f = %f.\n", x, y, fmod(x, y));

    return 0;
}
```

Output:
Remainder of -54.400000 / 10.000000 = -4.400000.

`fmodf`

`float` 型の値に関する `fmod()` 関数を実装します。[「fmod」\(p94\)](#) を参照してください。

`fmodl`

`long double` 型の値に関する `fmod()` 関数を実装します。[「fmod」\(p94\)](#) を参照してください。

`frexp` 実装

仮数 (mantissa) と指数を取り出します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double frexp(double value, int *exp);
float frexpf(float, int *);
long double frexpl(long double, int *);
```

引数 この関数の引数を以下に示します。

x	double、float、long double のいずれか	計算用の値
exp	int	指数

注意点 `frexp()` は式 $x \cdot 2^n$ の値の仮数と指数を取り出します。ここで、仮数は $0.5 \leq |x| < 1.0$ で、 n は整数の指数です。

戻り値 `frexp()` は `double` 型の仮数を返します。整数の指数の値は、`exp` で参照されるアドレスに記憶します。

参照 [「ldexp」\(p99\)](#)、[「fmod」\(p94\)](#)

例 16.10 `frexp()` の使用例

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double m, value = 12.0;
    int e;

    m = frexp(value, &e);

    printf("%f = %f * 2 to the power of %d.\n", value, m, e);

    return 0;
}
```

Output:
12.000000 = 0.750000 * 2 to the power of 4.

`frexpf`

`float` 型の値に関する `frexp()` を実装します。[「frexp 実装」\(p95\)](#) を参照してください。

frexpl

long double 型の値に関する frexp() を実装します。[「frexp 実装」\(p95 \)](#)を参照してください。

isgreater

2 つの doubles の大きい方を決定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
int isgreater(x, y)
```

引数 この関数の引数を以下に示します。

x	float、double、long double のいずれか	比較した数値
y	float、double、long double のいずれか	比較した数値

注意点 `x>y` `isgreater` と違い、`x` と `y` が丸められていないときも無効な例外を発生しません。

戻り値 `x` が `y` よりも大きければ真を返します。

isgreaterless

2 つの数値が等しくないかを調べます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
int isgreaterless(x, y)
```

引数 この関数の引数を以下に示します。

x	float、double、long double のいずれか	比較された数値
y	float、double、long double のいずれか	比較された数値

注意点 `x>y` || `x<y` `isgreaterless` と違い、`x` と `y` が丸められていないときも無効な例外を発生しません。

戻り値 `x` が `y` よりも大きい、または小さければ真を返します。

isless

2 つの数値の小さい方を調べます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
int isless(x, y)
```

引数 この関数の引数を以下に示します。

x	float、double、long double のいずれか	比較された数値
y	float、double、long double のいずれか	比較された数値

注意点 `x<y isless` と違い、`x` と `y` が丸められていないときも無効な例外を発生しません。

戻り値 `x` が `y` よりも小さければ真を返します。

islessequal

比較値よりも小さい、または等しいかをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
int islessequal(x, y)
```

引数 この関数の引数を以下に示します。

x	float、double、long double のいずれか	比較された数値
y	float、double、long double のいずれか	比較された数値

注意点 `x<y || x==y islessequal` と違い、`x` と `y` が丸められていないときも無効な例外を発生しません。

戻り値 `x` が `y` よりも小さい、または等しいときに真を返します。

isunordered

引数の順番を比較します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `int isunordered(x, y)`

引数 この関数の引数を以下に示します。

x	float、double、long double のいずれか	比較された数値
y	float、double、long double のいずれか	比較された数値

戻り値 引数が丸められていないときは真、それ以外の場合は偽を返します。

ldexp

仮数と指数から値を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double ldexp(double x, int exp);
float ldexpf(float, int);
long double ldexpl(long double, int);
```

引数 この関数の引数を以下に示します。

x	double、float、long double のいずれか	計算用の値
exp	int	指数

注意点 `ldexp()` 関数は $x * 2^{exp}$ を計算します。この関数は、`frexp()` で返された値から double 型の値を作る場合に利用できます。

戻り値 `ldexp()` は、 $x * 2^{exp}$ を返します。

参照 [「frexp 実装」\(p95\)](#)、[「modf」\(p102\)](#)

例 16.11 ldexp() の使用例

```
#include <math.h>
#include <stdio.h>

int main(void)
{
```

```
double value, x = 0.75;
int e = 4;

value = ldexp(x, e);

printf("%f * 2 to the power of %d is %f.\n", x, e, value);

return 0;
}
```

Output:
0.750000 * 2 to the power of 4 is 12.000000.

ldexpf

float 型の値に関する ldexp() 関数を実装します。[「ldexp」\(p99\)](#) を参照してください。

ldexpl

long double 型の値に関する ldexp() 関数を実装します。[「ldexp」\(p99\)](#) を参照してください。

log

自然対数を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <math.h>

```
double log(double x);
float logf(float);
long double logl(long double);
```

引数 この関数の引数を以下に示します。

x	double、float、long double のいずれか	計算される値
---	--------------------------------	--------

戻り値 log() は log_ex を返します。x < 0 である場合、log() は、errno に EDOM を割り当てます。

参照 [「インラインを積極的に利用する」\(p82\)](#) [「exp」\(p91\)](#) [「errno」\(p49\)](#)

例 16.12 log()、log10() の使用例

```
#include <math.h>
#include <stdio.h>
```

```
int main(void)
{
    double x = 100.0;

    printf("The natural logarithm of %f is %f\n",x, log(x));
    printf("The base 10 logarithm of %f is %f\n",x, log10(x));

    return 0;
}
```

Output:
The natural logarithm of 100.000000 is 4.605170
The base 10 logarithm of 100.000000 is 2.000000

logf

float 型の値に関する log() 関数を実装します。[「log」\(p100\)](#) を参照してください。

logl

long double 型の値に関する log() 関数を実装します。[「log」\(p100\)](#) を参照してください。

log10

常用対数を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <math.h>

```
double log10(double x);
float log10f(float);
long double log10l(long double);
```

引数 この関数の引数を以下に示します。

x	double、float、long double のいずれか	計算される値
---	--------------------------------	--------

戻り値 log10() は、log10x を返します。x < 0 log10() である場合、errno に EDOM を割り当てます。新しいエラーテスト手続きについては[「浮動小数点エラーテスト」\(p82\)](#) を参照してください。

参照 [「インラインを積極的に利用する」\(p82\)](#)、[「exp」\(p91\)](#)、[「errno」\(p49\)](#)

例 16.13 使用例は

[「log\(\), log10\(\) の使用例」\(p100\)](#) を参照してください。

log10f

float 型の値に関する log10() の関数を実装します。[「log10f」\(p101\)](#) を参照してください。

log10l

long double 型の値に関する log10() の関数を実装します。[「log10l」\(p101\)](#) を参照してください。

modf

整数部と小数部に分けます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double modf(double value, double *iptr);
float fmodf(float value, float *iptr);
long double modfl(long double value,
                  long double *iptr);
```

引数 この関数の引数を以下に示します。

value	double、float、long double のいずれか	分割する値
iptr	double、float、long double のいずれか	整数部

注意点 `modf()` 関数は、引数 `value` を整数部と小数部に分けます。言い換えると、`modf()` は `value = f + i` の `value` を取り出します。ここで、 $0 \leq f < 1$ 、`i` は `value` 以下の最大の値です。

戻り値 `modf()` は `value` の符号付きの小数部を返し、`iptr` で指し示される整数の部分にその整数を記憶します。

参照 [「frexp 実装」\(p95\)](#)、[「ldexp」\(p99\)](#)

例 16.14 `modf()` の使用例

```
#include <math.h>
#include <stdio.h>
```

```
int main(void)
{
    double i, f, value = 27.04;

    f = modf(value, &i);
    printf("The fractional part of %f is %f.\n", value, f);
    printf("The integer part of %f is %f.\n", value, i);

    return 0;
}
```

Output:
The fractional part of 27.040000 is 0.040000.
The integer part of 27.040000 is 27.000000.

modff

float 型の値に関する modf() 関数を実装します。[「modf」\(p102\)](#) を参照してください。

modfl

float 型の値に関する modf() 関数を実装します。[「modf」\(p102\)](#) を参照してください。

pow

xy を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <math.h>

```
double pow(double x, double y);
float powf(float x, float y);
long double powl(long double x, long double y);
```

引数 この関数の引数を以下に示します。

x	double、float、long double のいずれか	計算される値
---	--------------------------------	--------

戻り値 pow() は xy を返します。x が 0.0 でかつ y がゼロ以下である場合、または、x がゼロより小さくかつ y が整数でない場合、pow() 関数は errno に EDOM を割り当てます。新しいエラーテスト手続きについては[「浮動小数点エラーテスト」\(p82\)](#) を参照してください。

参照 [「インラインを積極的に利用する」\(p82\)](#)、[「sqrt」\(p106\)](#)、[「エラー検出の使用例」\(p83\)](#)

例 16.15 pow() の使用例

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double x;

    printf("Powers of 2:\n");
    for (x = 1.0; x <= 10.0; x += 1.0)
        printf("2 to the %4.0f is %4.0f.\n", x, pow(2, x));

    return 0;
}
```

Output:

```
Powers of 2:
2 to the    1 is    2.
2 to the    2 is    4.
2 to the    3 is    8.
2 to the    4 is   16.
2 to the    5 is   32.
2 to the    6 is   64.
2 to the    7 is  128.
2 to the    8 is  256.
2 to the    9 is  512.
2 to the   10 is 1024.
```

powf

float 型の値に関する pow() を実装します。[「pow」\(p103\)](#) を参照してください。

powl

long double 型の値に関する pow() を実装します。[「pow」\(p103\)](#) を参照してください。

sin

正弦 (sine) を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <math.h>


```
double sin(double x);  
float  sinf(float x);  
long double sinl(long double x);
```

引数 この関数の引数を以下に示します。

x	double、float、long double のいずれか	計算される値
---	--------------------------------	--------

注意点 `sin()` 関数の引数は、ラジアンです。1 ラジアンは、 $360/2$ 度です。

戻り値 `sin()` は、x の正弦を返します。x はラジアンとして評価されます。

参照 [「cos」\(p89\)](#)、[「tan」\(p108\)](#)

例 16.16 `sin()` の使用例

```
#include <math.h>  
#include <stdio.h>  
  
#define DtoR 2*pi/360  
  
int main(void)  
{  
    double x = 57.0;  
    double xRad = x*DtoR;  
  
    printf("The sine of %.2f degrees is %.4f.\n",x, sin(xRad));  
  
    return 0;  
}
```

Output:
The sine of 57.00 degrees is 0.8387.

`sinf`

float 型の値に関する `sin()` 関数を実装します。[「sin」\(p104\)](#) を参照してください。

`sinl`

long double 型の値に関する `sin()` 関数を実装します。[「sin」\(p104\)](#) を参照してください。

`sinh`

双曲線正弦 (hyperbolic sine) を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double sinh(double x);
float sinhf(float x);
long double sinhl(long double x);
```

引数 この関数の引数を以下に示します。

x	double、float、long double のいずれか	計算される値
---	--------------------------------	--------

戻り値 `sinh()` は、`x` の双曲線正弦を返します。

注意点 引数の絶対値が大きすぎる場合、範囲エラーが発生します。

参照 「インラインを積極的に利用する」(p82)、「cosh」(p90)、「tanh」(p109)

例 16.17 sinh() の使用例

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    double x = 0.5;
    printf("Hyperbolic sine of %f is %f.\n", x, sinh(x));

    return 0;
}
```

Output:
Hyperbolic sine of 0.500000 is 0.521095.

sinhf

float 型の値に関する sinh() を実装します。「sinh」(p105) を参照してください。

sinhl

long double 型の値に関する sinh() を実装します。[「sinh」\(p105\)](#)を参照してください。

sqrt

平方根を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double sqrt(double x);
float sqrtf(float x);
long double sqrtl(long double x);
```

引数 この関数の引数を以下に示します。

x double、float、long double のいずれか 計算用の値

戻り値 `sqrt()` は、x の平方根を返します。

注意点 引数が負の値の場合、領域エラーが発生します。

参照 [「インラインを積極的に利用する」\(p82\)](#)、[「pow」\(p103\)](#)

例 16.18 `sqrt()` の使用例

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double x = 64.0;

    printf("The square root of %f is %f.\n", x, sqrt(x));

    return 0;
}
```

Output:
The square root of 64.000000 is 8.000000.

`sqrtf`

float 型の値に関する `sqrt()` 関数を実装します。[「sqrt」\(p106\)](#) を参照してください。

`sqrtl`

long double 型の値に関する `sqrt()` 関数を実装します。[「sqrt」\(p106\)](#) を参照してください。

tan

正接 (tangent) を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double tan(double x);  
float tanf(float x);  
long double tanl(long double x);
```

引数 この関数の引数を以下に示します。

x double、float、long double のいずれか 計算用の値

戻り値 `tan()` は、x の正弦を返します。x はラジアンとして評価されます。

注意点 を 2 で割った値の奇数倍に引数が近づくと、範囲エラーを生じる可能性があります。

参照 [「インラインを積極的に利用する」\(p82\)](#)、[「cos」\(p89\)](#)、[「sin」\(p104\)](#)

例 16.19 tan() の使用例

```
#include <math.h>  
#include <stdio.h>  
  
int main(void)  
{  
    double x = 0.5;  
  
    printf("The tangent of %f is %f.\n", x, tan(x));  
  
    return 0;  
}
```

Output:
The tangent of 0.500000 is 0.546302.

tanf

float 型の値の `tan()` 関数を実装します。[「tan」\(p108\)](#) を参照してください。

tanh

long double 型の値の `tanh()` 関数を実装します。[「tanh」\(p108\)](#) を参照してください。

tanhf

双曲線正接 (hyperbolic tangent) を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
double tanh(double x);
float tanhf(float x);
long double tanhl(long double x);
```

引数 この関数の引数を以下に示します。

x	double、float、long double のいずれか	計算用の値
---	--------------------------------	-------

戻り値 `tanh()` は、x の双曲線正接を返します。

参照 [「cosh」\(p90\)](#)、[「sinh」\(p105\)](#)

例 16.20 tanh() の使用例

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double x = 0.5;

    printf("The hyperbolic tangent of %f is %f.\n", x, tanh(x));

    return 0;
}
```

Output:
The hyperbolic tangent of 0.500000 is 0.462117.

tanhf

float 型の値の `tanhf()` 関数を実装します。[「tanhf」\(p109\)](#) を参照してください。

tanhl

long double 型の値の tanh() 関数を実装します。[「tanh」\(p109\)](#) を参照してください。

HUGE_VAL

関数が返す、同符号で最大の浮動小数点の値です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

Varies by CPU

注意点 関数の結果が大きすぎて戻り値の型では表現できない場合、関数は HUGE_VAL を返します。これは期待された戻り値の型と同じ符号で、最大の浮動小数点の値となります。

C9X の実装

これらの math 関数は、ANSI/ISO 委員会によって公式に受理されたものではありませんが、いくつかのプラットフォームで既の実装されています。

acosh

acosh は、範囲 [0, +INF] の x の逆双曲線余弦 (arc hyperbolic cosine、負でない) を求めます。引数が 1 より小さい場合は領域エラーが発生し、x が大きすぎる場合は範囲エラーが発生します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`
`double acosh (double x);`

引数 この関数の引数を以下に示します。

x double 計算用の値

戻り値 x の双曲線余弦 (負でない) を返します。

参照 [「acos」\(p85\)](#)

asinh

逆双曲線正弦 (hyperbolic sine) を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`
`double asinh (double x);`

引数 この関数の引数を以下に示します。

x double 計算用の値

戻り値 引数 x の逆双曲線正弦を返します。

注意点 x の大きさがあまりにも大きい場合、範囲エラーが発生します。

参照 「[asin](#)」(p86)

atanh

関数 atanh は、逆双曲線正接 (arc hyperbolic tangent) を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`
`double atanh (double x);`

引数 この関数の引数を以下に示します。

x double 計算用の値

戻り値 x の逆双曲線正接を返します。

注意点 範囲 [-1,+1] に引数がない場合、領域エラーが発生します。

参照 「[atan](#)」(p86)

copysign

関数 copysign は、大きさ x と y の符号で 1 個の値を生成します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double copysign ( double x, double y );
```

引数 この関数の引数を以下に示します。

x	double	大きさ
y	double	符号用の引数

注意点 `copysign` 関数はゼロの符号を正と見なします。x が NaN の場合、y の符号付き NaN を生成します。

戻り値 x の大きさと y の符号を持った値を返します。

erf

関数 `erf` は、誤差関数を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double erf ( double x );
```

引数 この関数の引数を以下に示します。

x	double	計算される値
---	--------	--------

戻り値 x の誤差関数を返します。

erfc

補充誤差関数 (complementary error function) を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double erfc ( double x );
```


引数 この関数の引数を以下に示します。

x double 計算される値

戻り値 x の補充誤差関数を返します。

exp2

この関数は底が 2 の指数関数を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double exp2 ( double x );
```

引数 この関数の引数を以下に示します。

x double 計算用の値

戻り値 2 を底とした指数関数 $x:2^x$ を返します。

注意点 x が大きすぎる場合、範囲エラーが発生します。

参照 [「pow」\(p103\)](#)

expm1

関数 expm1 は、底が e の -1 指数関数を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double expm1 ( double x );
```

引数 この関数の引数を以下に示します。

x double 計算用の値

戻り値 底が 2 の x の指数関数から 1 を引いた値： $(e^x) - 1$ を返します。

注意点 x が大きすぎる場合、範囲エラーが発生します。x の大きさが小さい場合、`expm1(x)` は、`exp(x) - 1` より多くの正確さが要求されます。

fdim

関数 `fdim` は、引数の正の差分を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`
`double fdim (double x, double y);`

引数 この関数の引数を以下に示します。

x	double	値 1
y	double	値 2

戻り値 この関数は、y より x が大きい場合、x-y の値を返し、その他の場合、ゼロを返します。x が y 以下の場合、範囲エラーが発生するかもしれません。

fmax

関数 `fmax` は、その引数の最大数を求めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`
`double fmax (double x, double y);`

引数 この関数の引数を以下に示します。

x	double	第 1 引数
y	double	第 2 引数

戻り値 x と y の最大の値を返します。

参照 [「fmin」\(p114\)](#)

fmin

関数 `fmin` は、その引数の最小の値を求めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`
`double fmin (double x, double y);`

引数 この関数の引数を以下に示します。

x	double	第 1 引数
y	double	第 2 引数

戻り値 Fmin は、引数の最小値を返します。

参照 [「fmax」\(p114\)](#)

gamma

gamma 関数は、ガンマ関数を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`
`double gamma (double x);`

引数 この関数の引数を以下に示します。

x	double	計算される値
---	--------	--------

戻り値 x の gamma 関数を返します。

注意点 x がゼロと等しい、または負の整数である場合、領域エラーが発生します。範囲エラーが発生する可能性もあります。

hypot

関数 hypot は、引数の 2 乗の和の平方根を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`
`double hypot (double x, double y);`

引数 この関数の引数を以下に示します。

x	double	2 乗される 1 番目の値
y	double	2 乗される 2 番目の値

戻り値 x と y、それぞれ 2 乗の値の和の平方根を返します。

注意点 Hypot は、不当なオーバーフローあるいはアンダーフローなしに x と y の 2 乗の和の平方根を計算します。

- ・ 範囲エラーが発生する可能性があります。

参照 [「インラインを積極的に利用する」\(p82\)](#)

lgamma

関数 lgamma は値の絶対値の対数を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double lgamma ( double x );
```

引数 この関数の引数を以下に示します。

x	double	計算される値
---	--------	--------

戻り値 x のガンマの絶対値の対数を返します。

注意点 x が大きすぎる場合、範囲エラーが発生する可能性があります。

log1p

関数 log1p は自然対数を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double log1p ( double x );
```

引数 この関数の引数を以下に示します。

x	double	計算されている値
---	--------	----------

戻り値 x に 1 プラスした値の自然対数を返します。

注意点 x が小さい場合、log1p(x) は log(x+1) より多くの正確さが要求されます。

- ・ x が -1 より小さい場合、領域エラーが発生します。
- ・ x が 1 と等しい場合、範囲エラーが発生する可能性があります。

参照 [「log」\(p100\)](#)

log2

関数 log2 は、底が 2 の対数を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double log2 ( double x );
```

引数 この関数の引数を以下に示します。

x double 計算されている値

戻り値 x の底が 2 の対数を返します。

注意点 x がゼロより小さい場合、領域エラーが発生する可能性があります。x がゼロの場合、範囲エラーが発生する可能性があります。

参照 [「log」\(p100\)](#)

logb

関数 logb は、double 型の値の指数を取り出します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double logb ( double x );
```

引数 この関数の引数を以下に示します。

x double 計算されている値

戻り値 引数 x の形式での引数 x の指数（符号付き整数）を返します。

注意点 x が劣正規（subnormal）である場合、正規化されたものとして扱われます。 x がゼロと等しい場合、範囲エラーが発生する可能性があります。

参照 [「インラインを積極的に利用する」\(p82\)](#)

nan

関数 `nan` は、NaN をテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double nan( const char *tagp );
```

引数 この関数の引数を以下に示します。

tagp const char * 1 文字列

戻り値 可能である場合、NaN を返します。[「Quiet NaN」\(p81\)](#) を参照してください。

参照 [「isnan」\(p83\)](#)、[「NaN \(Not a Number\)」\(p81\)](#)

nearbyint

関数 `nearbyint` は、引数を整数値に四捨五入します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double nearbyint ( double x );
```

引数 この関数の引数を以下に示します。

x double 計算される値

戻り値 引数の中の整数値を返します。

注意点 `nearbyint` は `rint` に似ていますが、正確ではないことを表す例外処理は発生させません。

nextafter

関数 `nextafter` は y 方向の x 以降の、関数の型における次の表現可能な値を決定します。
ここで x と y は関数の型へ最初に変換されます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`

```
#define nextafter(x,y)
    ( (sizeof(x) == sizeof(float)) ?
      nextafterf(x,y) :
      (sizeof(x) == sizeof(double)) ?
      nextafterd(x,y)
```

引数 このマクロの引数を以下に示します。

x	float、double、long double	現在の表現可能な値
y	float、double、long double	方向

戻り値 x の後の次の表現可能な値を返します。

remainder

IEC559 によって要求された剰余 $x \text{ REM } y$ を求めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <math.h>`
`double remainder (double x , double y);`

引数 この関数の引数を以下に示します。

x	double	1 番目の値
y	double	2 番目の値

戻り値 $x \text{ REM } y$ の剰余を返します。

参照 [「remquo」\(p120\)](#)

remquo

関数 `remquo` は `remainder` 関数と同じ剰余を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double remquo (double x, double y, int *quo);
```

引数 この関数の引数を以下に示します。

x	double	1 番目の値
y	double	2 番目の値
quo	int*	商へのポインタ

戻り値 x の剰余と y を返します。

注意点 引数 `quo` は、次のような商を指します。その符号が `x/y` の符号で、その大きさは `x/y` の整数の商の大きさに対する `mod 2^n` と合同です。ここでは、`n>=3` です。

注意： `xx` の値は `y` に比例して `x` が大きくなる可能性があるので、商の正確な表現は現実的ではありません。

参照 [「remainder」\(p119\)](#)

rint

関数 `rint` は引数を整数値に四捨五入します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double rint ( double x );
```

引数 この関数の引数を以下に示します。

x	double	計算される値
---	--------	--------

戻り値 浮動小数点の形式での引数の整数の値を返します。

注意点 現在の丸めの方向を用いて、引数の整数値を浮動小数点形式で丸めます。

参照 [「rinttol」\(p121\)](#)

rinttol

関数 rinttol は、引数を最も近い long 型の整数値に丸めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
long int rinttol ( double x );
```

引数 この関数の引数を以下に示します。

x double 丸められている値

戻り値 浮動小数点の形式での引数の整数の値を返します。

注意点 Rintrol は、現在の丸めの方向を使って引数をもっとも近い整数値に丸めます。

- ・ 丸められた範囲が long 型の範囲外の場合、結果は指定されません。

参照 [「rint」\(p120\)](#)

round

引数を浮動小数点の形式で整数の値に丸めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double round ( double x );
```

引数 この関数の引数を以下に示します。

x double 丸め用の値

戻り値 浮動小数点の形式での引数の丸められた、引数以下の整数値を返します。

注意点 現在の丸めの方向に関わらず、整数と整数の間の値をゼロから遠い方に丸めます。

参照 [「roundtol」\(p122\)](#)

roundtol

関数 roundtol は引数を最も近い整数値に丸めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
long int roundtol ( double round );
```

引数 この関数の引数を以下に示します。

round double 丸められた値

戻り値 引数は int フォーマットの整数値に丸められます。

注意点 現在の丸めの方向に関わらず、整数と整数の間の値をゼロから遠い方に丸めます。

- 丸められた範囲が long 型の範囲外である場合、結果は指定されていません。

参照 [「round」\(p121\)](#)

scalb

関数 scalb は $x * FLT_RADIX^n$ を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double scalb ( double x, long int n );
```

引数 この関数の引数を以下に示します。

x double 元となる値
n long int ベキ乗の値

戻り値 $x * FLT_RADIX^n$ を返します。

注意点 関数 scalb は FLT_RADIX^n を直接計算せず、効果的に $x * FLT_RADIX^n$ を計算します。

- 範囲エラーが発生する可能性があります。

trunc

trunc は引数を浮動小数点の形式で整数の値に丸めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <math.h>
double trunc ( double x );
```

注意： 68k プロセッサは整数の値を返します。

引数 この関数の引数を以下に示します。

x double 切り捨てられる値

戻り値 浮動小数点形式での引数の整数値を返します。

注意点 浮動小数点形式で引数より大きくならない最も近い整数値に丸めます。

第 17 章 path2fss.h

path2fss.h ヘッダファイルは PBMakeFSSpec と同様の関数、path2fss を宣言します。

path2fss.h の概要

path2fss.h ヘッダファイルは以下のものから構成されます。

[「path2fss」\(p125\)](#) : PBMakeFSSpec と同様の関数

path2fss

この関数は PBMakeFSSpec と同様です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <path2fss.h>
OSErr __path2fss
    (const char * pathName, FSSpecPtr spec)
```

引数 この関数の引数です。

pathname	const char *	パス名
spec	FSSpecPtr	ファイル指定ポインタ

注意点 この関数は 3 つの大きな違いはあるものの、PBMakeFSSpec と同様です。

入力 (C の文字列) としてパス名のみが用いられ、引数ブロックは用いられません。

ファイルに関して FSSpecs が作成され、ディレクトリは作成されません。

HFS をサポートする任意のシステムバージョンでの、* 任意 * の HFS Mac (Mac 512KE、Mac Plus あるいはそれ以降) で動作します。

MFS ディスクを正確に扱います (63 文字以上のファイル名では割り込みを生じ、bdNamErr を返す)。

PBMakeFSSpec のように、この関数は指定されたファイルが存在せず、FSSpec が新しいファイルを生成する目的で有効である場合、fnfErr を返します。

戻り値 無効なパス名、あるいはファイルではなくディレクトリを指定したパス名では、エラーを返します。

参照 『Inside Macintosh : Files』

第 18 章 Process.h



Process.h ヘッダファイルでは、threadex 関数、_beginthreadex と _endthreadex が宣言されています。

Process.h の概要

Process.h ヘッダファイルは以下から構成されます。

[「_beginthreadex」\(p127 \)](#)

[「_endthreadex」\(p128 \)](#)

_beginthreadex

スレッドの開始を示します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

```
プロトタイプ #include <process.h>
HANDLE __cdecl _beginthreadex(
    LPSECURITY_ATTRIBUTES inSecurity,
    DWORD inStacksize,
    LPTHREAD_START_ROUTINE inCodeAddress,
    LPVOID inParameter,
    DWORD inCreationFlags,
    LPDWORD inThreadID);
```

引数 この関数に関する引数を以下に示します。

inSecurity	LPSECURITY_ATTRIBUTES	セキュリティ属性 (NULL がデフォルトの属性)
inStacksize	DWORD *	リンカの /STACK スイッチを設定 (1MB がデフォルト)
inCodeAddress	LPTHREAD_START_ROUTINE	新しいスレッドが開始するコードを持った関数のアドレス
inParameter	LPVOID	通常 lpvThreadParameter として渡されるものと同じで、初期化ルーチンに渡すために利用する

inCreationFlags	DWORD	ゼロの場合、スレッドを直ちに開始、CREATE_SUSPENDED の場合、実行前に待機する
inThreadID	LPDWORD	新しいメソッドへ割り当てられた ID を格納する変数

戻り値 成功したとき HANDLE 変数を返します。

注意点 関数 `_beginthreadex` は Windows 呼び出しの `CreateThread` と似ています。ただし、この関数は MSL によって使われるローカルデータを生成する点がそれとは異なっています。

参照 [「_endthreadex」\(p128\)](#)

`_endthreadex`

スレッドの終了を示します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <process.h>`
`VOID __cdecl _endthreadex(DWORD inReturnCode);`

引数 この関数に関する引数を以下に示します。

`inReturnCode` `DWORD` 終了コードはこの引数を通して渡される

戻り値 なし。スレッドを終了します。

注意点 関数 `_beginthreadex` は Windows 呼び出しの `ExitThread` と似ています。ただし、この関数は MSL によって使われるスレッドのローカルデータを削除する点が異なっています。

参照 [「_beginthreadex」\(p127\)](#)

第 19 章 setjmp.h

setjmp.h ヘッダファイルはプロセッサ状態の保存と復元の方法を提供します。

setjmp.h の概要

setjmp.h ヘッダファイルの関数は、プログラミングエラーや下位レベルの割り込みハンドラにおいて一般的に利用されます。

関数「[setjmp](#)」([p130](#)) は、現在の呼び出し環境（現在のプロセッサの状態）を引数 jmp_buf に保存します。jmp_buf の型は、配列でプロセッサのプログラムカウンタ、スタックポインタ、関連のあるデータレジスタ、アドレスレジスタを保存します。

関数「[longjmp](#)」([p129](#)) は、最後に setjmp() が呼ばれた時点での状態にプロセッサを復元させます。言い換えれば、setjmp() と longjmp() のペアが引数として同じ jmp_buf 変数を使っている場合、longjmp() は最後の setjmp() が呼ばれたところへプログラムの実行を戻します。

非ローカルジャンプと例外の取り扱い

jmp_buf 変数は大域変数となりうるので、setjmp と longjmp の呼び出しは同じ関数の本体でなくてもかまいません。

jmp_buf 変数は、longjmp() が使用される前に、setjmp() の呼び出しによって初期化されている必要があります。初期化されていない jmp_buf 変数を使った longjmp() の呼び出しは、プログラムをクラッシュさせる可能性があります。コンパイラの最適化を使ってレジスタに割り当てられた変数は、setjmp() と longjmp() の呼び出しの間の実行中に変更を受ける可能性があります。この状態は、影響を受けた変数を volatile 型として宣言することによって避けることができます。

longjmp

setjmp() で保存されたプロセッサの状態を復元します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <setjmp.h>
void longjmp(jmp_buf env, int val);
```

引数 この関数の引数です。

env	jmp_buf	現在のプロセッサの状態
val	int	setjmp() が返した値

注意点 longjmp() 関数は、呼び出した環境を env 変数を使った最後の setjmp() の呼び出しで保存された状態に復元します（つまり、プログラムの実行を復帰させます）。プログラムの実行は、setjmp() 関数から続けられます。プロセッサの状態が復元された時の引数 val は、setjmp() によって返された値です。

警告！ Longjmp 関数は、Altivec をサポートする際に再定義されました。「to」と「from」のコンパイルユニットの両方で Altivec が有効になっていることを確かめてください。有効にしていない場合、予期せぬ結果が生じます。

戻り値 longjmpが終了した後、対応するsetjmpマクロ呼び出しがvalによって指定された値を戻したかのように、プログラムの実行は続けられます。longjmp 関数は setjmp マクロに値 0 を戻すように指定することはできません。値が 0 であれば、setjmp マクロは 1 を戻します。

警告！ env 変数は、プログラムの実行中での好ましくない結果を避けるため、longjmp() で使う前に、setjmp() によって初期化する必要があります。

参照 [「setjmp」\(p130\)](#)、[「signal」\(p135\)](#)、[「abort」\(p244\)](#)

例 19.1 longjmp() の使用例

[「setjmp\(\) の例」\(p131\)](#)

setjmp

longjmp() のためのプロセッサの状態を保存します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <setjmp.h>
int setjmp(jmp_buf env);

引数 この関数の引数です。

env	jmp_buf	現在のプロセッサの状態
-----	---------	-------------

注意点 setjmp() 関数は、呼び出し環境 - データレジスタ、アドレスレジスタ、スタックポインタ、プログラムカウンタ - を引数 env に保存します。この引数は、longjmp() に渡される前に setjmp() で初期化する必要があります。

警告！ Longjmp 関数は、Altivec をサポートする際に再定義されました。「to」と「from」のコンパイルユニットの両方で Altivec が有効になっていることを確かめてください。有効にしていなければ、予期せぬ結果が生じます。

戻り値 最初に呼び出しが生じた場合、setjmp() は、プロセッサの状態を保存し、0 を返します。longjmp() が呼び出されると、setjmp() が env に保存したプロセッサの状態へプログラムの実行を移します。longjmp() への呼び出しを通じて setjmp() が起動される場合、setjmp() は longjmp() の val を返します。

参照 [「longjmp」\(p129\)](#)、[「signal」\(p135\)](#)、[「abort」\(p244\)](#)

例 19.2 setjmp() の例

```
#include <setjmp.h>
#include <stdio.h>
#include <stdlib.h>

// Let main() と doerr() 双方とも
// 大域変数 env へアクセスします。
volatile jmp_buf env;

void doerr(void);
int main(void)
{
    int i, j, k;

    printf("Enter 3 integers that total less than 100.\n");
    printf("A zero sum will quit.\n\n");

    // 入力された数が 100 以下でない場合、
    // プログラムの実行はこの地点から再スタートします。

    if (setjmp(env) != 0)
        printf("Try again, please.\n");

    do {
        scanf("%d %d %d", &i, &j, &k);
        if ( (i + j + k) == 0)
            exit(0); // プログラム終了
        printf("%d + %d + %d = %d\n\n", i, j, k, i+j+k);
        if ( (i + j + k) >= 100)
```

```
        doerr(); // エラー!
    } while (1); // 無限ループ

    return 0;
}

void doerr(void) // これはエラーの処理
{
    printf("The total is >= 100!\n");
    longjmp(env, 1);
}

Output:

Enter 3 integers that total less than 100.
A zero sum will quit.

10 20 30
10 + 20 + 30 = 60

-4 5 1000
-4 + 5 + 1000 = 1001

The total is >= 100!
Try again, please.
0 0 0
```

第 20 章 signal.h

signal.h ヘッダファイルには、ソフトウェアの割り込みの仕様が列挙されています。

signal.h の概要

シグナルはソフトウェア割り込みです。プログラムの中止、浮動小数点に関する例外、異常命令の割り込み、シグナル化されたユーザ割り込み、セグメント違反、プログラムの終了に関するシグナルがあります。BeOS 用に追加されたセマンティックホールドの実装は、[「Be 特有のシグナルの取り扱い」\(p137\)](#) を参照してください。

[「signal.h シグナルの説明」\(p134\)](#): signal.h ファイルでマクロとして定義されているシグナルについて

[「signal」\(p135\)](#): シグナルの取り扱い方法（シグナルの無視、通常の方法での取り扱い、プログラマが提供したシグナル処理関数を使った方法）

[「raise」\(p137\)](#): シグナルの処理関数を呼び出す

[「シグナル関数の処理引数」\(p135\)](#): 関数を拡張するために、あらかじめ定義されたシグナル処理マクロについて

BeOS では以下の機能が追加されます。

[「sigaction」\(p139\)](#): シグナルの動作

[「sigprocmask」\(p139\)](#): 処理手順のマスク

[「sigpending」\(p140\)](#): シグナルの保留

[「sigsuspend」\(p140\)](#): シグナルの一時中止

[「kill」\(p141\)](#): シグナルの強制終了

[「send_signal」\(p141\)](#): シグナルの送信

[「struct vregs」\(p141\)](#): シグナル処理用の構造体

シグナル処理

シグナルは、raise() 関数を使って呼び出されます。シグナルが生じた場合、関連の関数が実行されます。

CodeWarrior C の signal.h では、シグナルは関数 [「raise」\(p137\)](#) を使ってのみ呼び出すことができ、SIGABRT シグナルは関数 [「abort」\(p244\)](#) を使って呼び出すことができます。シグナルが発生すると、通常の間数呼び出しでシグナル処理関数が実行されます。

SIGTERM を除く全てのシグナルのデフォルトのシグナルハンドラは SIG_DFL です。SIGTERM シグナルは `exit()` を使ったプログラムを正常終了させるのに対して、SIG_DFL シグナルは `abort()` 関数を使ったプログラムを中止します。

ANSI C の標準ライブラリでは、`signal.h` での名前の前に SIG と付くマクロは、将来の利用のために予約されていると明記されています。プログラマは標準ライブラリの将来の仕様と合致しないように名前の前に SIG を使うことは避けるべきです。

`signal.h` の `typedef char sig_atomic_t` 型は、非同期割り込みの間、汚染のない構成要素としてアクセスすることが可能です。

シグナルの数は、このヘッダ内の `__signal_max` の値によって定義されます。

警告! `printf()`、`getchar()`、`malloc()` のようなプロテクトされていない再入可能関数を使って、シグナルハンドラからハードウェアに対してシグナルを発生させるのは、いかなるシステムでも推奨できません。シグナルは効果的な割り込みであり、すでに関数内にあっても任意の場所で発生します。マルチスレッドでは、関数がそれ自身を再入からプロテクトする場合、シグナルハンドラから再入したときに失敗することがあります。

表 20.1 signal.h シグナルの説明

マクロ	説明
SIGABRT	中止シグナル。このマクロは正の整数値として定義される。このシグナルは <code>abort()</code> 関数によって呼び出される。
SIGFPE	浮動小数点例外シグナル。このマクロは正の整数値として定義される。
SIGILL	不当命令割り込み。このマクロは正の整数値として定義される。
SIGINT	対話型ユーザ割り込みシグナル。このマクロは、正の整数値として定義される。
SIGSEGV	セグメント違反シグナル。このマクロは正の整数値として定義される。
SIGTERM	終了シグナル。このマクロは正の整数値として定義される。このシグナルが発生すると、 <code>exit()</code> 関数を呼び出すことによってプログラムを終了させる。

`signal()` 関数はシグナルの処理方法（シグナルの無視、通常の方法での取り扱い、プログラムによる取り扱い（シグナル処理関数の提供））を指定します。[「シグナル関数の処理引数」\(p135\)](#) では、予め定義されたシグナル処理マクロについて説明します。

表 20.2 シグナル関数の処理引数

マクロ	説明
SIG_IGN	このマクロは、void を返す関数へのポインタに変更する。シグナルを無視する場合、signal() 中の関数引数で利用する。
SIG_DFL	このマクロは、void を返す関数へのポインタに変更する。このシグナルハンドラは、オープンしたストリームのフラッシュやクローズすることなしにプログラムを終了する。
SIG_ERR	マクロの SIG_IGN と SIG_DFL は関数ポインタとして定義される。この値は、signal() に渡された要求を受け取ることができない場合に返される。

signal

シグナル処理を設定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <signal.h>
void (*signal(int sig, void (*func)(int)))(int);
```

引数 この関数の引数です。

sig	int	シグナル処理関数に対応する番号
func	void *	シグナル処理関数へのポインタ

注意点 signal() 関数は、int 型の値の引数を得るシグナル処理ルーチンへのポインタを返します。

引数 sig はシグナル処理関数に対応したシグナル番号です。signal.h で定義されたシグナルは「[signal.h シグナルの説明](#)」(p134) で説明します。

引数 func はシグナル処理関数です。この関数はプログラマが提供したものか、「[シグナル関数の処理引数](#)」(p135) で説明した予め定義されたシグナルハンドラのいずれかです。

シグナルが発生した場合、signal(sig, SIG_DFL) を呼び出すことによって、シグナルハンドラの実行を優先させます。この signal() の呼び出しはユーザのハンドラを効果的に無効にします。ユーザハンドラ内の signal() に対する呼び出しをその関数引数のユーザハンドラとすることで、再登録することができます。

戻り値 signal() は、シグナルsigに関する signal() への最後の呼び出しによって設定されたシグナルハンドラ関数へのポインタを返します。

参照 「[raise](#)」(p137)、 「[abort](#)」(p244)、 「[atexit](#)」(p246)、 「[exit](#)」(p256)

例 20.1 signal() の使用例

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

void userhandler(int);

void userhandler(int sig)
{
    char c;

    printf("userhandler!\nPress return.\n");

    /* リターンキーが押されるまで待機 */
    c = getchar();
}

int main(void)
{
    void (*handlerptr)(int);
    int i;

    handlerptr = signal(SIGINT, userhandler);
    if (handlerptr == SIG_ERR)
        printf("Can't assign signal handler.\n");

    for (i = 0; i < 10; i++) {
        printf("%d\n", i);
        if (i == 5) raise(SIGINT);
    }

    return 0;
}
```

Output:

```
0
1
2
3
4
5
userhandler!
Press return.

6
7
8
9
```

raise

シグナルを発生します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <signal.h>`
`int raise(int sig);`

引数 この関数の引数です。

sig int シグナル処理関数

注意点 `raise()` 関数は、シグナル `sig` に対応するシグナル処理関数を呼び出します。

戻り値 `raise()` はシグナルが成功した場合ゼロを返し、失敗した場合非ゼロを返します。

参照 [「longjmp」\(p129\)](#)、[「signal」\(p135\)](#)、[「abort」\(p244\)](#)、[「atexit」\(p246\)](#)、[「exit」\(p256\)](#)

例 20.2 `raise()` の使用例

[「signal\(\) の使用例」\(p136\)](#) の例を参照してください。

Be 特有のシグナルの取り扱い

シグナル処理関数に関しての Posix インタフェースは、使い易いものではありません。標準ではシグナルハンドラに渡されるのは単独の引数 (シグナル番号) です。多くの情報を持つことは有用であり、BeOS では 2 つの拡張引数を提供しています。

表 20.3 BeOS 固有のシグナルの定義

マクロ	説明
SIGHUP	停止 - tty が反応しなくなった
SIGQUIT	tty でタイプされた quit 特別文字
SIGCHLD	子プロセスが終了された
SIGPIPE	パイプ w/no リードへ書き込む
SIGKILL	チームの強制終了 (キャッチ不可能)
SIGSTOP	スレッドの休止 (キャッチ不可能)
SIGCONT	休止である場合、実行を継続

マクロ	説明
SIGTSTP	tty にタイプされた stop の特殊文字
SIGALRM	アラームの開始 (alarm() を参照)
SIGTTIN	bg プロセスから tty の読み込み
SIGTTOU	bg プロセスから tty への書き込み
SIGUSR1	app 定義のシグナル 1
SIGUSR2	app 定義のシグナル 2
SIGKILLTHR	指定するチームではなくスレッドを強制終了

「[BeOS シグナル関数処理引数](#)」(p138) では、あらかじめ定義された BeOS のシグナル処理マクロについて説明します。

表 20.4 BeOS シグナル関数処理引数

マクロ	説明
sigemptyset	セットを空にする
sigfillset	セットを一杯にする
sigaddset	セットを追加する
sigdelset	セットを削除する
sigismember	メンバ

BeOS では、`__signal_func_ptr` 型としての `sigaction struct` の `sa_handler` フィールドを宣言します。これは、`sa_handler` フィールドへ割り当てられるいかなる関数もキャストが必要であることを意味します。

注意： C++ メンバ関数は、シグナルハンドラであってはいけません。(なぜなら、最初の引数として `this` ポインタを想定しているからです)

BeOS がシグナルハンドラに対して提供する 3 引数を以下に示します。

- 最初の引数はシグナル番号 (1 個の整数型)
- 次の引数は `sigaction struct` の `sa_userdata` フィールドに置かれる値の全て
- 最後の引数は `vregs struct` へのポインタ

`vregs struct` は、シグナルがスレッドへ引き渡されたときのレジスタの内容です。構造体の内容は変更することができます。シグナルハンドラが終了した後、OS はスレッドに関して、この `struct` をレジスタへ再ロードする際に利用します (特権レジスタにはロードされません)。 `vregs struct` は非常にマシンに依存しており、PowerPC ファミリーの異なったモデルの間でさえも変更が約束されています。それを利用する場合、新しいプロセッ

サが出現したとき、プログラムを再び動作させなければならないことを予想すべきです。それにも関わらず、レジスタを変更する機能はいくつかの興味あるプログラミングの可能性を切り開きます。

```
struct sigaction {
    __signal_func_ptr sa_handler;
    sigset_t          sa_mask;
    int               sa_flags;
    void              *sa_userdata;
    /* シグナルハンドラへ受け渡す */
};
```

表 20.5 BeOS シグナルフラグ

マクロ	説明
SIG_NOCLDSTOP	sa_flags に関する
SIG_BLOCK	sigprocmask() の how arg に関して定義する
SIG_UNBLOCK	非ブロック化する
SIG_SETMASK	マスクを設定する

sigaction

関数 sigaction はシグナルの動作です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <signal.h>
int sigaction(
    int sig,
    const struct sigaction *act,
    struct sigaction *oact);
```

引数 この関数の引数を以下に示します。

sig	int	シグナル
act	const struct sigaction *	シグナル動作
oact	struct sigaction *	シグナル動作

戻り値 整数値を返します。

sigprocmask

関数 sigprocmask はシグナル処理手順のマスクです。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <signal.h>
int sigprocmask(
    int how,
    const sigset_t *set,
    sigset_t *oset);
```

引数 この関数の引数を以下に示します。

how	int	方法
set	const sigset_t *	セット
oset	sigset_t *	0 セット

戻り値 整数値を返します。

sigpending

関数 sigpending はシグナル休止に関するものです。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <signal.h>
int sigpending(sigset_t *set);
```

引数 この関数の引数を以下に示します。

set	sigset_t *	セット
-----	------------	-----

戻り値 整数値

sigsuspend

関数 sigsuspend は休止したシグナルを示します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <signal.h>
int sigsuspend(const sigset_t *mask);
```

引数 この関数の引数を以下に示します。

mask const sigset_t * シグナルセットマスク

戻り値 整数値

kill

関数 kill はシグナルを終了させます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <signal.h>
int kill(pid_t pid, int sig);
```

引数 この関数の引数を以下に示します。

pid pid_t
sig int

戻り値 整数値

send_signal

関数 send_signal はシグナルを送出します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <signal.h>
int send_signal(pid_t tid, uint sig);
```

引数 この関数の引数を以下に示します。

tid pid_t
sig uint

戻り値 整数値を返します。

struct vregs

シグナルハンドラは最後の引数としてこれを取得します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <signal.h>
typedef struct vregs
{
    ulong pc,                /* プログラムカウンタ */
    r0,                      /* スクラッチ */
    1,                        /* スタック ptr */
    r2,                      /* TOC */
                                /* 揮発性 regs */
    r3, r4,r5,r6,r7,r8,r9,r10,
                                /* スクラッチ regs */
    r11, r12;

    double f0,                /* fp スクラッチ */

                                /* fp 揮発性 regs */
    f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13;

    ulong filler1,            /* 位置ホルダー */
    fpscr,                    /* fp 状態コード */
    ctr, xer, cr, msr, lr;    /* さまざまな状態 */
}vregs;
```

第 21 章 SIOUX と WinSIOUX

SIOUX と WinSIOUX (Simple Input and Output User eXchange) ライブラリはメニュー、ウィンドウ、イベントなどのグラフィカルユーザーインターフェースを処理します。

SIOUX と WinSIOUX の概要

Mac OS 版のインターフェースは SIOUX、Windows 版のインターフェースは WinSIOUX と呼ばれます。SIOUX (Standard Input Output User eXchange) コンソールのインタフェースと構造体のメンバを以下に示します。

[「SIOUX と WinSIOUX を使う」\(p143\)](#) : SIOUX の属性の説明

[「Windows 用 WinSIOUX」\(p144\)](#) : Windows 95/NT 用の SIOUX ライブラリ

[「Mac OS 用 SIOUX」\(p146\)](#) : Mac OS オペレーティングシステム用の SIOUX ライブラリ

SIOUX と WinSIOUX を使う

DOS や UNIX などのコマンドラインインターフェースで記述されたプログラムを移植したり、完全なグラフィカルユーザーインターフェース (ウィンドウ、メニュー、イベントなど) を書く時間がないのに、新しいプログラムを書かなくてはならないかもしれません。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

そのような作業を支援するために、CodeWarrior は SIOUX および WinSIOUX ライブラリを提供しています。これらはウィンドウ、メニュー、タイトルなどのグラフィカルユーザーインターフェース項目を処理するので、プログラムで処理する必要はありません。ダムターミナルやスクロール可能な TTY に似たウィンドウを作成します。標準 C 関数や C++ オペレータ (`printf()`、`scanf()`、`getchar()`、`putchar()`) や、C++ の `inserter`、`extractor` オペレータ (`<<` and `>>`) で読み取りや書き込みが可能です。SIOUX および WinSIOUX ライブラリはウィンドウの内容の保存や印刷を行うファイルメニューも作成します。Mac OS 版の SIOUX はウィンドウの内容のカット、コピー、ペーストを行う Edit メニューも作成します。Mac OS 上では `stdin`、`stdout`、`cout`、`cin` input output やコマンドライン引数のリダイレクトも可能です。

参照 [「console.h の概要」\(p25\)](#)

注意： UNIX や DOS のプログラムを移植する場合、他のヘッダの互換性のある関数が必要になるかもしれません。

Windows 用 WinSIOUX

WinSIOUX ウィンドウはサイズ変更とスクロールが可能なテキストウィンドウです。ここでプログラムがテキストの読み取りと書き込みを行います。

ファイルメニューのコマンドでSIOUXウィンドウの内容を保存、印刷することができます。

[「WinSIOUX を使うプロジェクトを作成」\(p144\)](#)：WinSIOUX プログラムの作成手順

[「WinSIOUX のカスタマイズ」\(p144\)](#)：WinSIOUX コンソール作成の設定

[「WinSIOUXclrscr」\(p145\)](#)：WinSIOUX コンソールスクリーンとバッファのクリア

[「clrscr」\(p146\)](#)：WinSIOUX コンソールスクリーンとバッファのクリア

WinSIOUX を使うプロジェクトを作成

WinSIOUX ライブラリを使うには、WinSIOUX Console 形式のプロジェクトステーションナリからプロジェクトを作成してください。

WinSIOUX プロジェクトには少なくとも以下のライブラリを必ず追加してください。

ANSICx86.LIB

ANSICx86sd.LIB

注意： 今回のバージョンの WinSIOUX は完全ではありません。

WinSIOUX のカスタマイズ

`tSIOUXBuffer` 構造体を修正することによって WinSIOUX 環境をカスタマイズする方法を説明します。WinSIOUX は `tSIOUXBuffer` のデータフィールドを調べて WinSIOUX ウィンドウと環境を作成する方法を決定します。

注意： WinSIOUX をカスタマイズするには、標準の入力、出力を使う関数を呼び出すよりも先に `tSIOUXBuffer` を修正しなくてはなりません。呼び出しの後で `tSIOUXBuffer` を修正しても、WinSIOUX のウィンドウは変化しません。

表 21.1 tSIOUXBuffer 構造体

型	要素	概要
char *	startpos	バッファとして使われるメモリブロックへのポインタ
char *	curtop	スクリーン上部の行頭へのポインタ
char *	endpos	バッファ内のテキストの終点へのポインタ
char *	inputstart	キーボード入力バッファとして使われるメモリブロックへのポインタ
char *	inputcur	次に利用可能な入力文字へのポインタ
char *	inputlast	最終入力文字の後ろの文字へのポインタ
char *	SelBasePtr	選択が始まった位置（開始点または終点）へのポインタ
char *	SelStartPtr	選択テキストの開始点へのポインタ
char *	SelEndPtr	選択テキストの終点へのポインタ
int	row	現在の挿入位置の行インデックス
int	maxrow	行の最大数
int	col	現在の挿入位置の列インデックス
int	maxcol	列の最大数
int	installed	コンソールがインストールされていれば真
int	inputavail	プログラムで入力が可能ならば真
int	dirtybit	最後に保存したときからバッファが変更されていれば真
int	NeedInput	キャレット位置で文字が保存されれば真
int	CmdShow	
int	numlines	バッファ内のテキスト行の数
long	bufsize	在のバッファサイズ

WinSIOUXclrscr

WinSIOUX ウィンドウをクリアし、バッファをフラッシュします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <WinSIOUX.h>`
`void WinSIOUXclrscr(void);`

引数 なし。

注意点 この関数はコンソールと WinSIOUX バッファをクリアするために使われます。

参照 [「SIOUXclrscr」\(p153\)](#)

clrscr

WinSIOUX ウィンドウをクリアし、バッファをフラッシュします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <WinSIOUX.h>
void clrscr(void);
```

引数 なし。

注意点 WinSIOUXclrscr を呼び出します。

参照 [「WinSIOUXclrscr」\(p145\)](#)

Mac OS 用 SIOUX

Mac OS 用の SIOUX には以下のセグメントが含まれます。

[「SIOUX を用いたプロジェクトの作成」\(p147\)](#): 実行中の SIOUX プログラムを表示

[「SIOUX のカスタマイズ」\(p148\)](#): SIOUX ウィンドウのカスタマイズ方法

- ・ [「SIOUXSettings 構造体」\(p148\)](#): SIOUX の外観を変更するための構造体メンバ

[「ユーザーアプリケーションでの SIOUX ウィンドウの使用」\(p152\)](#): SIOUX プロジェクトで Mac OS 機能を利用

- ・ [「SIOUXHandleOneEvent」\(p154\)](#): SIOUX でイベントを使用
- ・ [「SIOUXSetTitle」\(p155\)](#): SIOUX ウィンドウにカスタムタイトルを指定

注意: SIOUX に基づく Marco Piovaneli の WASTE はプレリリースバージョンが入手可能です。これは 32k 以上の文字をスクリーンに出力できます。通常の SIOUX 関数は動作しますが、注意が必要です。リリースノートをお読みください。

ウィンドウはサイズ変更とテキストウィンドウのスクロールが可能で、ユーザーのプログラムはそこでテキストを読み書きすることが可能です。プログラムのテキストは 32K まで保存することができます。

編集メニューのコマンドを使って、テキストを SIOUX ウィンドウからカットしたりコピーすることができ、他のアプリケーションから SIOUX ウィンドウへペーストしたりできま

す。ファイルメニューのコマンドを使って、SIOUX ウィンドウの内容を印刷や保存することができます。

プログラムを任意の時間で停止させるには、Command + . (ピリオド) か Control + C を押します。SIOUX アプリケーションは実行し続けるので、ウィンドウの内容を編集したり保存することができます。プログラムが終了した時点で終了したい場合、またはダイアログウィンドウを保存するかどうかの質問を表示されたくない場合は、[「終了時の動作の変更」\(p152\)](#) を参照してください。

SIOUX アプリケーションを終了させるには、ファイルメニューの終了を選択してください。ウィンドウの内容が保存されていない場合、アプリケーションはウィンドウの内容を保存するかどうかを尋ねるダイアログを表示します。状態を表す行を削除したい場合 [「状態を表す行の表示」\(p152\)](#) を参照してください。

SIOUX を用いたプロジェクトの作成

SIOUX ライブラリを使うためには、プロジェクトステーションリを使ってコンソール型のプロジェクトを生成してください。

注意： この章では、標準入力と標準出力は、stdin、stdout、cin、cout を参照します。stderr、clog、cerr への標準エラーの出力は ccommand() を使ってファイルヘリダイレクトできません。

標準入力からの読み込み、または出力への書き込みがある場合、特別な関数や特別なヘッダファイルをインクルードする必要はありません。プログラムが標準入力や出力を参照した場合、SIOUX ライブラリは自動的に実行を開始し SIOUX ウィンドウを生成します。

注意： printf() や scanf() のような関数は、シンボルが引数リストに現れなくても標準入力と出力を使います。

SIOUX の環境をカスタマイズしたい場合、標準入力や出力を利用する前に SIOUX.h をインクルードして、SIOUXSettings を変更してください。使用した後、SIOUX はウィンドウを生成して変更が不可能となります。詳細は [「SIOUX のカスタマイズ」\(p148\)](#) を参照してください。

イベントループを持つプログラム中で SIOUX ウィンドウを使いたい場合、SIOUXSettings を修正して、関数 SIOUXHandleOneEvent() を呼び出してください。詳細は [「ユーザーアプリケーションでの SIOUX ウィンドウの使用」\(p152\)](#) を参照してください。

すでに生成されたプロジェクトに SIOUX を加えたい場合、関連のライブラリを含める必要があります。

68K のプロジェクトは少なくとも以下のライブラリを含む必要があります。

MSL SIOUX.68K.Lib

MacOS.Lib
MSL Runtime68k.lib
68k プロジェクトに適した MathLib
68k プロジェクトに適した MSL .C.Lib

PPC プロジェクトは、少なくとも以下のライブラリを含む必要があります。

MSL SIOUX.PPC.Lib
InterfaceLib
MSL RuntimePPC.lib
MathLib
MSL C.PPC.Lib

SIOUX のカスタマイズ

構造体 SIOUXSettings を変更することによって、SIOUX の環境のカスタマイズする方法について説明します。SIOUX は、SIOUX のウィンドウと環境をどのように生成するか決定するために、SIOUXSettings のデータフィールドを調べます。

注意： SIOUX をカスタマイズするために、標準入力あるいは出力を利用する任意の関数を呼び出す前に SIOUXSettings を修正してください。後から SIOUXSettings を修正した場合、SIOUX はウィンドウを変更しません。

最初の 3 つの節、[「フォントとタブの変更」\(p150\)](#)、[「サイズと位置の変更」\(p151\)](#)、[「状態を表す行の表示」\(p152\)](#) では、SIOUX ウィンドウのカスタマイズの方法について説明します。次の節 [「終了時の動作の変更」\(p152\)](#) では、終了時の SIOUX の動作の修正の方法を説明します。最後の節 [「ユーザーアプリケーションでの SIOUX ウィンドウの使用」\(p152\)](#) では、Mac OS プログラム上での SIOUX ウィンドウを使う方法について説明します。

[「SIOUXSettings 構造体」\(p148\)](#) では、SIOUXSettings 構造体について概説します。

表 21.2 SIOUXSettings 構造体

構造体のフィールド		仕様
char	initializeTB	Macintosh Toolbox の初期化の有無
char	standalone	ユーザー自身のイベントループを利用するか SIOUX のイベントループを利用するかを選択
char	setupmenus	アプリケーションに対してファイルと編集のメニューを生成するか否を選択
char	autocloseonquit	ユーザープログラムが完了した場合に自動的にアプリケーションを終了するか否かを選択

構造体のフィールド		仕様
char	asktosaveonclose	プログラムの動作が完了した場合に SIOUX の出力をファイルに保存するか否かを質問
char	showstatusline	SIOUX ウィンドウの状態を表示する行を表示するか否かを選択
short	tabspace	ゼロより大きい場合、タブをその数だけ空白で置き換え、ゼロの場合、タブを出力
short	column	SIOUX ウィンドウの 1 行の文字数
short	rows	SIOUX ウィンドウのテキストの行数
short	toppixel	SIOUX ウィンドウの 1 番上の位置 x
short	leftpixel	SIOUX ウィンドウの左の位置
short	fontid	SIOUX ウィンドウのフォント
short	fontsize	SIOUX ウィンドウのフォントのサイズ
short	fontface	SIOUX ウィンドウのフォントのスタイル

「[SIOUX ウィンドウをカスタマイズする例](#)」(p149) に、SIOUX ウィンドウをカスタマイズする小さなプログラムがあります。

例 21.1 SIOUX ウィンドウをカスタマイズする例

```
#include <stdio.h>
#include <sioux.h>
#include <MacTypes.h>
#include <Fonts.h>

int main(void)
{
    short familyID;

    /* Don't exit the program after it runs or ask whether
       to save the window when the program exit */
    SIOUXSettings.autocloseonquit = false;
    SIOUXSettings.asktosaveonclose = false;

    /* Don't show the status line */
    SIOUXSettings.showstatusline = false;

    /* Make the window large enough to fit 1 line
       of text that contains 12 characters. */
    SIOUXSettings.columns = 12;
    SIOUXSettings.rows = 1;

    /* Place the window's top left corner at (5,40). */
    SIOUXSettings.toppixel = 40;
```

```
SIOUXSettings.leftpixel = 5;

/* Set the font to be 48-point, bold, italic Times.    */
SIOUXSettings.fontsize = 48;
SIOUXSettings.fontface = bold + italic;
GetFNum("\ptimes", &familyID);
SIOUXSettings.fontid = familyID;

printf("Hello World!");

return 0;
}
```

フォントとタブの変更

この節では、SIOUX が `tabspace` フィールドを用いて、タブの処理を変更する方法について説明します。また、`fontid`、`fontsize`、`fontface` フィールドを用いてフォントの変更方法について説明します。

注意： SIOUX ウィンドウの状態を表す行には、`fontid`、`fontsize`、`fontface` フィールドで指定されたフォントを使ったメッセージが書き込まれます。フォントが大きすぎた場合、状態を表す行は読めないかもしれません。[「状態を表す行の表示」\(p152\)](#)で説明する状態を合わす行は、`showstatusline` フィールドを偽に設定することによって削除することができます。

SIOUX ウィンドウのフォントを変更するには、`fontid` を以下の値のいずれかに設定してください。

```
courier の ID : kFontIDCourier
geneva の ID : kFontIDGeneva
helvetica の ID : kFontIDHelvetica
monaco の ID : kFontIDMonaco
newYork の ID : kFontIDNewYork
symbol の ID : kFontIDSymbol
times の ID : kFontIDTimes
```

デフォルトでは、`fontid` は、`monaco` です。

フォントに関する文字のスタイルを変更するには、以下の値のいずれかを `fontface` に設定します。

```
normal
bold
```

```
italic  
underline  
outline  
shadow  
condense  
extend
```

スタイルの組み合わせを行うには、別のスタイルを追加してください。例えば **bold** と *italic* のテキストを書くには、`fontface` を `bold + italic` に設定します。デフォルトでは `fontface` は `normal` です。

フォントのサイズを変更するには、`fontsize` にそのサイズを設定します。デフォルトでは、`fontsize` は 9 ポイントです。

`tabspaces` フィールドは、SIOUX がどのようにタブを処理するかを制御します。`tabspaces` が 0 より大きな値である場合、SIOUX はタブの代わりにその数だけ空白を書き出します。`tabspaces` が 0 である場合、タブを出力します。SIOUX ウィンドウでは、1 つのタブは 1 つのスペースのように見えます。表を印刷する場合、`tabspaces` を 4 や 8 の適当な数に設定してください。デフォルトは 4 です。

以下のサンプルでは、フォントが 12 ポイント、**bold**、*italic* New York、タブは 4 つの空白という設定です。

```
SIOUXSettings.fontsize = 12;  
SIOUXSettings.fontface = bold + italic;  
SIOUXSettings.fontid = kFontIDNewYork;  
SIOUXSettings.tabspaces = 4;
```

サイズと位置の変更

SIOUX では、SIOUX ウィンドウのサイズと位置を変更ができます。

ウィンドウのサイズを変更するには、`rows` にウィンドウのテキストの行数を設定し、`columns` にそれぞれの行の文字数を設定します。SIOUX は、`fontid`、`fontsize`、`fontface` で指定されたフォントを調べて、ユーザーが設定した行と文字の数を含むのに十分なウィンドウを生成します。ウィンドウがモニタに入りきれないぐらい大きな場合、SIOUX はモニタに表示可能な大きさのウィンドウを生成します。

例えば、以下の設定は 1 行に 40 文字、10 行のウィンドウを生成します。

```
SIOUXSettings.rows = 10;  
SIOUXSettings.columns = 40;
```

デフォルトの SIOUX ウィンドウは 1 行 80 文字 24 行です。

SIOUX ウィンドウの位置を変更するには、ウィンドウを置きたい場所を指すように `toppixel` と `leftpixel` を設定します。`toppixel` と `leftpixel` は、SIOUX ウィンドウの左上隅を指しています。`toppixel` を 38、`leftpixel` を 0 とすることで、ウィンドウをメニューバー直下の可能な限り左に置くことができます。`toppixel` が 38 以下なら SIOUX

ウィンドウはメニューバーの下に入ります。toppixel と leftpixel が両方とも 0 である場合、SIOUX はその場所にウィンドウを置かず、代わりにモニタの中央に置きます。

例えば以下の設定は、ウィンドウをメニューバーの真下でモニタの左端の近くに置きます。

```
SIOUXSettings.toppixel = 40;  
SIOUXSettings.leftpixel = 5;
```

終了時の動作の変更

フィールド autocloseonquit と asktosaveonclose は、プログラムが終了して SIOUX がそのウィンドウを閉じる時に、SIOUX の動作を決定できます。

フィールド autocloseonquit では、プログラムの実行が終了した時点で SIOUX の動作を決定できます。autocloseonquit が真である場合、SIOUX は自動的に終了します。autocloseonquit が偽である場合、SIOUX は実行を続け、終了するにはファイルメニューから終了を選択しなければなりません。デフォルトでは autocloseonquit は偽です。

ヒント： SIOUX ウィンドウの内容はファイルメニューの保存を選択していつでも保存することができます。

asktosaveonclose フィールドは、SIOUX の終了時にその動作を決定できます。asktosaveonclose が真である場合、SIOUX は SIOUX ウィンドウの内容を保存するかどうかを尋ねるダイアログを表示します。asktosaveonclose が偽である場合、SIOUX はダイアログを表示することなしに終了します。デフォルトでは asktosaveonclose は真です。

例えば以下の設定では、ユーザープログラムが終了したとき、出力の保存を質問せずに SIOUX アプリケーションが終了します。

```
SIOUXSettings.autocloseonquit = TRUE;  
SIOUXSettings.asktosaveonclose = FALSE;
```

状態を表す行の表示

showstatusline フィールドは、SIOUX ウィンドウが状態を表す行の表示するかどうかを制御します。これはプログラムが実行中、出力中、入力待ちであるという情報が含まれています。showstatusline が真である場合、状態を表す行が表示されます。showstatusline が偽である場合、状態を表す行は表示されません。デフォルトでは showstatusline は偽です。

ユーザーアプリケーションでの SIOUX ウィンドウの使用

ここでは SIOUX のユーザープログラムの制御について説明します。最初は、ユーザープログラムで SIOUX がどのように動作するかを理解する必要があります。SIOUX 環境は、他の関数として main() 関数を呼ぶ 1 つのアプリケーションとして考えられます。SIOUX が main() を呼ぶ前に Macintosh Toolbox とそのメニューを準備するように初期化を行います。main() が終了した後、SIOUX は生成したものの片付けを行います。main() 実行中で間で

さえ、SIOUX はそれが入力中か出力中かの場合はいつでも、選択したメニューや押されたコマンドキーで動作します。

また、SIOUX はそれがどの程度動作するかを選択することができます。ユーザー自身のイベント、メニュー、Macintosh Toolbox の初期化を行うことが可能です。

ユーザー自身のイベントを処理するアプリケーションを記述し、簡単な入出力の SIOUX ウィンドウを利用したい場合、標準入出力を利用する前にフィールド `standalone` を偽に設定してください。SIOUX は自身のイベントループを使わず、フィールド `autocloseonquite` を真に設定して、ユーザープログラムが終了すると同時にアプリケーションを終了します。ユーザーのイベントループでは、[「ユーザーアプリケーションでの SIOUX ウィンドウの使用」\(p152\)](#) に記述されている関数 `SIOUXHandleOneEvent()` を呼び出す必要があります。

SIOUX のメニューを使いたくなければ、`setupmenus` フィールドを偽に設定します。`standalone` も偽である場合、メニューを生成せず、ユーザープログラムには何も起こりません。`standalone` が真である場合、ユーザー自身のメニューを生成して操作することができます。

ユーザー自身の Macintosh Toolbox を初期化したい場合、`initializeTB` フィールドを偽に設定します。`standalone` フィールドは `initializeTB` に影響を与えません。

例えば以下の行は、ユーザー自身のイベントの処理、メニューの生成、Macintosh Toolbox の初期化を行うアプリケーションに対する SIOUX の設定です。

```
SIOUXSettings.standalone = false;
SIOUXSettings.setupmenus = false;
SIOUXSettings.initializeTB = false;
```

SIOUXclrscr

SIOUX ウィンドウをクリアし、バッファをフラッシュします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <SIOUX.h>`
`void SIOUXclrscr(void);`

引数 なし。

注意点 この関数はコンソールと SIOUX バッファをクリアするために使われます。

参照 [「WinSIOUXclrscr」\(p145\)](#)

SIOUXHandleOneEvent

SIOUX ウィンドウに対するイベントの処理です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <SIOUX.h>`
`Boolean SIOUXHandleOneEvent(EventRecord *event);`

注意点 イベントを処理する前に、`SIOUXHandleOneEvent()` を呼び出してください。これは、必要な場合、SIOUX がウィンドウを更新できます。引数 `event` は、`WaitNextEvent()` や `GetNextEvent()` が返した値でなければなりません。イベントを処理した場合、関数は真を返し、処理していなければ偽を返します。イベントが NULL ポインタである場合、イベントを受け取るまで関数はイベントをポーリングします。

戻り値 これがイベントを処理する場合、`SIOUXHandleOneEvent()` は真を返します。それ以外の場合、`SIOUXHandleOneEvent()` は偽を返します

例 21.2 SIOUXHandleOneEvent() の使用例

```
void MyEventLoop(void)
{
    EventRecord event;
    RgnHandle cursorRgn;
    Boolean gotEvent, SIOUXDidEvent;

    cursorRgn = NewRgn();

    do {
        gotEvent = WaitNextEvent(everyEvent, &event,
                                MyGetSleep(), cursorRgn);

        /* ユーザー自身のイベントを処理する前に、
         * イベントが SIOUX に関するものかどうかを調べるために、
         * SIOUXHandleOneEvent() を呼び出す
         */
        if (gotEvent)
            SIOUXDidEvent = SIOUXHandleOneEvent(&event);

        if (!SIOUXDidEvent)
            DoEvent(&event);

    } while (!gDone)
}
```

SIOUXSetTitle

SIOUX 出力ウィンドウのタイトルを設定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
include <SIOUX.h>
extern void SIOUXSetTitle
(unsigned char title[256])
```

引数 この関数の引数です。

title unsigned char [] Pascal 文字列

注意点 SIOUX ウィンドウに対する出力の後、SIOUXSetTitle() を呼び出す必要があります。関数 SIOUXSetTitle() は、タイトルが設定されなければエラーを返します。コンソールに対する書き込みは、改行が書き込まれる、ストリームがフラッシュされる、プログラムが終了するなどの動作が発生するまで実行されません。

警告！ SIOUXSetTitle() に関する引数は Pascal 型の文字列で、C 型の文字列ではありません。

戻り値 SIOUXSetTitle() からの戻り値はありません。

例 21.3 SIOUXSetTitle() の使用例

```
#include <stdio.h>
#include <SIOUX.h>

int main(void)
{
    printf("Hello World\n");
    SIOUXSetTitle("\pMy Title");

    return 0;
}
```


第 22 章 stat.h

stat.h ヘッダファイルは、UNIX からプログラムを移植するための便利ないくつかの関数を宣言しています。

stat.h の概要

stat.h には以下の関数があります。

「[Stat 構造体と定義](#)」: stat 構造体と型

「[fstat](#)」(p158): オープンしたファイルの情報を取得

「[mkdir](#)」(p160): フォルダとしてディレクトリを作成

「[stat](#)」(p161): ファイルの統計量を取得

stat.h と UNIX の互換性

ヘッダファイル stat.h は、UNIX からプログラムを移植するための便利な関数をいくつか宣言しています。これらの関数は多くの UNIX ライブラリの関数と同様です。しかし、UNIX と Mac OS オペレーティングシステムは基本的に異なり同一ではありません。関数のそれぞれの説明では違いについて説明しています。

一般的に、新しいプログラムではこれらの関数を使いたくない場合、代わりに Macintosh Toolbox の代用のプログラムを使います。

注意: UNIX や DOS プログラムを移植する場合、UNIX と互換性のあるヘッダが必要になるかもしれません。

Stat 構造体と定義

stat.h ヘッダは stat 構造体、型の定義、モード定義を含んでいます。

表 22.1 定義済みの型

型	... を保存する
nlink_t	リンクの数
uid_t	ユーザーの ID

型	... を保存する
gid_t	ファイルサイズ
off_t	ファイルサイズのバイト

表 22.2 Stat 構造体

型	変数	用途
mode_t	st_mode	ファイルモード（参照： 「ファイルモード」 ）
ino_t	st_ino	ファイルのシリアル番号
dev_t	st_dev	このファイルを含むデバイスの ID
nlink_t	st_nlink	リンクの数
uid_t	st_uid	ファイル所有者のユーザー ID
gid_t	st_gid	ファイルグループの Group ID
off_t	st_size	ファイルサイズのバイト数
time_t	st_atime	最後にアクセスした時間
time_t	st_mtime	最後にデータを修正した時間
time_t	st_ctime	最後にファイル状態を変更した時間

表 22.3 ファイルモード

ファイルモード	用途
S_IFMT	ファイルタイプのマスク
S_IFDIR	ディレクトリ
S_IFCHR	特殊文字
S_IFIFO	パイプ
S_IFREG	レギュラー
S_IREAD	読み取り許可、所有者
S_IWRITE	書き込み許可、所有者
S_IEXEC	実行 / 検索許可、所有者

fstat

用途 オープンしたファイルについての情報を取得します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stat.h>`
`int fstat(int fildes, struct stat *buf);`

引数 この関数の引数です。

fildes	int	ファイルの説明
buf	struct stat *	stat 構造体のアドレス

注意点 関数は `fildes` に関連するファイルの情報を得て、`buf` が指す構造体へその情報を格納します。構造体には「[Stat 構造体](#)」([p158](#)) に挙げるフィールドがあります。

戻り値 成功した場合 `fstat()` はゼロを返します。失敗した場合、`fstat()` は `-1` を返し `errno` を設定します。

参照 「[stat](#)」([p161](#))

例 22.1 fstat() の使用例

```
#include <stdio.h>
#include <time.h>
#include <unix.h>

int main(void)
{
    struct stat info;
    int fd;

    fd = open("mytest", O_WRONLY | O_CREAT | O_TRUNC);
    write(fd, "Hello world!\n", 13);

    fstat(fd, &info);
    /* オープンしたファイルの情報を取得する */

    printf("File mode:          0x%lX\n", info.st_mode);
    printf("File ID:            0x%lX\n", info.st_ino);
    printf("Volume ref. no.:       0x%lX\n", info.st_dev);
    printf("Number of links:      %hd\n", info.st_nlink);
    printf("User ID:              %lu\n", info.st_uid);
    printf("Group ID:             %lu\n", info.st_gid);
    printf("File size:            %ld\n", info.st_size);
    printf("Access time:          %s", ctime(&info.st_atime));
    printf("Modification time:    %s", ctime(&info.st_mtime));
    printf("Creation time:         %s", ctime(&info.st_ctime));

    close(fd);

    return 0;
}
This program may print the following:
```

File mode:	0x800
File ID:	0x5ACA
Volume ref. no.:	0xFFFFFFFF
Number of links:	1
User ID:	200
Device type:	0
File size:	13

mkdir

用途 フォルダを作成します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stat.h>
int mkdir(const char *path, int mode); /* Mac */
int mkdir(const char *path); /* Windows */
```

引数 この関数の引数です。

path	const char *	パス名
mode	int	オープンモード (Windows では使用できない)

注意点 この関数は path で指定された新しいフォルダを生成します。引数 mode は無視されます。

戻り値 成功した場合、mkdir() はゼロを返します。エラーである場合、mkdir() は、-1 を返し、errno を設定します。

参照 [「unlink」\(p335 \)](#)、[「rmdir」\(p333 \)](#)

例 22.2 mkdir() の使用例

Macintosh

```
#include <stdio.h>
#include <stat.h>

int main(void)
{
    if( mkdir(":Akbar", 0) == 0)
        printf("Folder Akbar is created");

    return 0;
}
```


Windows

```
#include <stdio.h>
#include <stat.h>

int main(void)
{
    if( mkdir(".\Akbar") == 0)
        printf("Folder Akbar is created");

    return 0;
}
```

Creates a folder named Akbar as a sub-folder of the current folder

stat

用途 ファイルについての情報を取得します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <stat.h>

```
int stat(const char *path, struct stat *buf);
```

引数 この関数の引数です。

path	const char *	パス名
buf	struct stat *	stat 構造体へのポインタ

注意点 この関数は path で指定したファイルの情報を得て、buf で指された構造体へその情報を格納します。構造体には「[Stat 構造体](#)」(p158) 説明するフィールドがあります。

戻り値 成功した場合 stat() はゼロを返します。

参照 「[fstat](#)」(p158)、 「[uname](#)」(p349)

例 22.3 stat() の使用例

```
#include <stdio.h>
#include <time.h>
#include <unix.h>
```

```
int main(void)
{
    struct stat info;

    stat("Akbar:System Folder:System", &info);
    /* システムファイルの情報を取得する */

    printf("File mode:           0x%lX\n", info.st_mode);
    printf("File ID:             0x%lX\n", info.st_ino);
    printf("Volume ref. no.:        0x%lX\n", info.st_dev);
    printf("Number of links:        %hd\n", info.st_nlink);
    printf("User ID:                %lu\n", info.st_uid);
    printf("Group ID:               %lu\n", info.st_gid);
    printf("File size:              %ld\n", info.st_size);
    printf("Access time:            %s", ctime(&info.st_atime));
    printf("Modification time:      %s", ctime(&info.st_mtime));
    printf("Creation time:          %s", ctime(&info.st_ctime));

    return 0;
}
```

This program may print the following:

```
File mode:           0x800
File ID:             0x4574
Volume ref. no.:     0x0
Number of links:     1
User ID:             200
Group ID:            100
File size:           30480
Access time:         Mon Apr 17 19:46:37 1995
Modification time:   Mon Apr 17 19:46:37 1995
Creation time:       Fri Oct  7 12:00:00 1994
```

第 23 章 stdarg.h

stdarg.h ヘッダファイルは、引数の数を変化させる関数の生成を可能とします。

stdarg.h の概要

可変引数に関する stdarg.h での機能を以下に示します。

[「va_arg」\(p163\)](#): 可変引数リスト

[「va_end」\(p164\)](#): 可変引数の終わり

[「va_start」\(p164\)](#): 可変引数の始まり

関数の可変引数

stdarg.h ヘッダファイルは引数の数を変化させることができる関数の生成を許可します。

可変長引数の関数は、その最後の引数として省略 (...) を使って定義します。例えば、

```
int funnyfunc(int a, char c, ...);
```

関数は va_list 型、va_start()、va_arg()、va_end() マクロを使って記述します。

関数は、関数引数のリストを保持するように宣言された va_list 変数を持ちます。マクロ [「va_start」\(p164\)](#) は、va_list 変数を初期化して引数のアクセスが始まる前に呼び出されます。マクロ [「va_arg」\(p163\)](#) は va_list の引数のそれぞれを返します。すべての引数が va_list を通じて処理されると、マクロ [「va_end」\(p164\)](#) が関数からの正常の戻り値を与えるために呼び出されます。

va_arg

引数の値を返すマクロです。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdarg.h>`
`type va_arg(va_list ap, type);`

引数 この関数の引数です。

ap	va_list	変数リスト
type	type	マクロがセットしたタイプ

注意点 va_arg() マクロは、関数の引数リスト上の次の引数を返します。返された引数は、type によって定義された型を持ちます。引数 ap は、va_start() マクロによって最初に初期化される必要があります。

戻り値 va_arg() マクロは、関数の type 型引数リストでの次の引数を返します。

参照 [「va_end」\(p164\)](#)、[「va_start」\(p164\)](#)

例 23.1 va() の例

[「va_start\(\) の使用例」\(p165\)](#) の例を参照してください。

va_end

正常時の関数の戻り値を準備します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdarg.h>
void va_end(va_list ap);
```

引数 この関数の引数です。

ap	va_list	変数リスト
----	---------	-------

注意点 va_end() 関数は、関数の戻り値を与えるためにスタックを整理します。関数は、関数引数が va_arg() マクロを使ってアクセスされた後、呼び出されます。

参照 [「va_arg」\(p163\)](#)、[「va_start」\(p164\)](#)

例 23.2 va_end の使用例

[「va_start\(\) の使用例」\(p165\)](#) の例を参照してください。

va_start

可変長引数リストを初期化します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdarg.h>`
`void va_start(va_list ap, ParmN);`

引数 この関数の引数です。

ap	va_list	変数リスト
Parm	ParmN	最後に名前を付けた引数

注意点 `va_start()` マクロは、初期化を行い引数リストを `ap` に割り付けます。引数 `ParmN` は関数の原型において、省略 (...) の前の最後に名付けられた引数です。

参照 [「va_arg」\(p163\)](#)、[「va_end」\(p164\)](#)

例 23.3 `va_start()` の使用例

```
#include <stdarg.h>
#include <string.h>
#include <stdio.h>

void multisum(int *dest, ...);

int main(void)
{
    int all;

    all = 0;
    multisum(&all, 13, 1, 18, 3, 0);
    printf("%d\n", all);

    return 0;
}

void multisum(int *dest, ...)
{
    va_list ap;
    int n, sum = 0;

    va_start(ap, dest);

    while ((n = va_arg(ap, int)) != 0)
        sum += n; /* 次の引数を dest に加える */
    *dest = sum;
    va_end(ap); /* 立ち去る前の物事の整理 */
}
```

Output :
35

第 24 章 stddef.h

stddef.h ヘッダファイルは、ANSI C の標準ライブラリで共通に使用されるマクロと型を定義します。

stddef.h の概要

stddef.h では、共通に利用されるマクロと型を定義します。

[「NULL」\(p167\)](#): NULL の定義

[「offsetof」\(p167\)](#): 構造体メンバのオフセット

[「ptrdiff_t」\(p167\)](#): ポインタの差

[「size_t」\(p168\)](#): sizeof 命令からの戻り値

[「wchar_t」\(p168\)](#): ワイドキャラクタ型

共通に利用される定義

stddef.h ヘッダファイルは、ANSI C の標準ライブラリで共通に使用されるマクロと型を定義しています。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

NULL

NULL マクロは、標準ライブラリで利用されるヌルポインタ定数です。

offsetof

offsetof(構造体、メンバ)マクロは size_t 型の整数表現を拡張します。返された値は、構造体の基底からメンバのオフセットのバイト数です。

注意： メンバが bit フィールドである場合、結果は定義されません。

ptrdiff_t

ptrdiff_t 型は、あるポインタの値からもう一方を引く場合に利用される符号付き整数型です。

`size_t`

`size_t` 型は、`sizeof()` 命令によって返された符号付き整数型の値です。

`wchar_t`

`wchar_t` 型は、ASCII 文字セットの全部の文字表現を保持することができる整数の型です。

第 25 章 stdio.h

stdio.h ヘッダファイルは、入出力制御に関する関数を提供します。

stdio.h の概要

stdio.h ヘッダファイルには、ファイルの生成、削除、名前の変更、ランダムアクセスの許可、テキスト、バイナリファイルの書き込み、読み出しに関する関数があります。

stdio.h ヘッダで提供される機能です。

- [「clearerr」\(p172\)](#): ストリームからのエラーをクリアする
- [「fclose」\(p173\)](#): ファイルをクローズする
- [「fdopen」\(p175\)](#): ファイル記述子をストリームに変換する
- [「ferror」\(p177\)](#): ファイルエラーの状態をチェックする
- [「fflush」\(p179\)](#): ストリームをフラッシュする
- [「fgetc」\(p180\)](#): ファイルから 1 文字読み込む
- [「fgetpos」\(p181\)](#): ファイル上での位置を取得する
- [「fgets」\(p183\)](#): ファイルから文字列を読み込む
- [「fopen」\(p184\)](#): ファイルをオープンする
- [「fprintf」\(p187\)](#): ファイルへ書式化された出力を行う
- [「fputc」\(p191\)](#): ファイルへ 1 文字を書き込む
- [「fputs」\(p193\)](#): ファイルへ文字列を書き込む。
- [「fread」\(p194\)](#): ファイルを読み込む
- [「freopen」\(p196\)](#): ファイルをリダイレクトでオープンする
- [「fscanf」\(p197\)](#): ファイルをスキャンする
- [「fseek」\(p201\)](#): ファイル上での位置を移動させる
- [「fsetpos」\(p204\)](#): ファイル上での位置を設定する
- [「ftell」\(p204\)](#): ファイル上での位置を知らせる
- [「fwrite」\(p205\)](#): ファイルへ書き込む
- [「getc」\(p206\)](#): ストリームから 1 文字読み込む
- [「getchar」\(p208\)](#): 標準入力 (stdin) から 1 文字読み込む

[「gets」\(p209\)](#): 標準入力 (stdin) から文字列を読み込む

[「perror」\(p210\)](#): 標準エラー出力 (stderr) へエラーを書き込む

[「printf」\(p211\)](#): 標準出力 (stdout) へ書式化された出力を書き込む

[「putc」\(p217\)](#): ストリームへ 1 文字書き込む

[「putchar」\(p218\)](#): 標準出力 (stdout) へ 1 文字書き込む

[「puts」\(p219\)](#): 標準出力 (stdout) へ文字列を書き込む

[「remove」\(p220\)](#): ファイルを削除する

[「rename」\(p221\)](#): ファイルの名前を変更する

[「rewind」\(p222\)](#): ファイル指標をファイルの始まりにリセットする

[「scanf」\(p224\)](#): 入力を標準入力としてスキャンする

[「setbuf」\(p228\)](#): ストリームのバッファサイズを設定する

[「setvbuf」\(p229\)](#): ストリームのバッファサイズとバッファリングの計画を設定する

[「sprintf」\(p231\)](#): 文字バッファに書き込む

[「sscanf」\(p232\)](#): 文字列をスキャンする

[「tmpfile」\(p233\)](#): テンポラリファイルを生成する

[「ungetc」\(p235\)](#): ストリーム上で 1 文字後退する

[「vfprintf」\(p237\)](#): 可変な引数リストを利用してファイルに書き込む

[「vprintf」\(p239\)](#): 可変な引数リストを利用して標準出力 (stdout) に書き込む

[「vsprintf」\(p240\)](#): 可変な引数リストを利用して文字配列に書き込む

標準入力 / 出力

ストリーム

ストリームとはファイルを抽象化したもので、ハードウェアの I/O リクエストを減少させるために設計されました。バッファリングがない場合、I/O デバイス上のデータは一度に 1 項目しかアクセスできません。accessed この非効率的な I/O の処理では、プログラムの実行をかなり遅くしてしまいます。stdio.h の関数は、ファイルに読み書きするデータを取り込んだり、集めるために主記憶中のバッファを利用します。ファイルへの読み書きでバッファが一杯になった場合、I/O アクセスの回数を減少させます。バッファの内容は、fflush() 関数を用いて早めにファイルに送ることが可能です。

stdio.h ヘッダは、3 つのバッファリング計画 (バッファ化なし、ブロックバッファ化、ラインバッファ化) を提供しています。setvbuf() 関数は、任意の出力ストリームのバッファリング計画を変更するために使われます。出力ストリームをバッファしない場合、送ったデータを直ちにファイルへ読み書きします。

出力ストリームがブロックバッファ化の場合、データを主記憶のバッファに蓄えます。バッファが一杯になった場合、その内容を目的のファイルに送り、バッファそのものをクリアします。さらに、ストリームがクローズされるまでその処理を繰り返します。

ラインバッファ化された出力ストリームは、ブロックバッファ化された出力ストリームと同様に動作します。異なる点は、データをバッファ内で集めますが、改行の文字（'\n'）で行が終わらなければファイルに送らないことです

ストリームは FILE へのポインタを用いて宣言されます。3 つの FILE ポインタ、FILE *stdin、FILE *stdout、FILE *stderr を自動的にプログラム用にオープンします。FILE ポインタ stdin と stdout は、標準入力と出力ファイルで、それぞれ、対話型のコンソール I/O です。stderr ファイルポインタは、標準エラー出力ファイルで、エラーメッセージをコンソールに書き込みます。stdin、stdout、stderr ストリームは、ラインバッファ化されています。

stdin、stdout、stderr を Mac OS のコンソールウィンドウに表示することに関する情報は、SIOUX.h の章を参照してください。

ファイル位置指標

ファイル位置指標は、stdio.h ヘッダで導入された概念です。オープンされたそれぞれのストリームは、ファイル中でカーソルのように動くファイル位置指標を持っています。ファイル位置指標は、読み書きするための次の文字の位置を指しており、一度の読み書き動作が、ファイル位置指標を進ませます。関数によっては、ファイルに対して読んだり書いたりすることなしに指標を調整することが可能で、これによりファイルに対してランダムアクセスが提供されます。

コンソールストリームの stdin、stdout、stderr は特別で、ファイル位置指標を持たないことに注意してください。

ファイルの終わりとエラー

stdio.h で定義された多くの関数は、EOF の値を返すストリームから読み込みを行います。EOF の値は、最後の読み書きでファイルの終わりに到達したところを示す非ゼロの値です。

いくつかの stdio.h の関数は、errno 大域変数を使用しているので、errno.h ヘッダの章を参照してください。errno の使用は、以下の関数の説明の中で説明します。

拡張文字とバイト文字のストリーム方向

入力と出力には 2 種類のストリーム方向があります。拡張文字 (wchar_t) 方向とバイト (char) 方向です。ストリームがファイルと関連づけられた後は、ある操作が行われるまでストリームに方向はなくなります。

ストリームで何らかの操作が行われると、その操作によってストリームに拡張文字またはバイトの方向が生じます。その方向はファイルを閉じて再度開くまで残ります。

ストリームの方向が確立された後、他の方向の関数呼び出しはできません。つまりバイト方向の I/O 関数は、拡張文字方向のストリームでは効果がありません。

ストリーム方向と標準の入力 / 出力

あらかじめ定義されている 3 つのストリーム (`stdin`、`stdout`、`stderr`) は、プログラムの起動時には方向が決まっていません。標準の入力 / 出力ストリームが閉じていると、コンソールへのそのストリームを再度開いて接続することはできません。しかし、名前のあるファイルへのストリームを再度開いて接続することはできます。

C/C++ の入力 / 出力関数は同じ `stdin`、`stdout`、`stderr` ストリームを共有します。

clearerr

ストリーム中のファイルの終わりとエラーの状態をクリアします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
void clearerr(FILE *stream);
```

引数 この関数の引数です。

stream FILE * FILE ストリームへのポインタ

注意点 `clearerr()` 関数は、ストリームのファイルの終わりの状態とエラー状態をリセットします。ストリームをオープンしたときにもこれらの状態をリセットします。

参照 [「feof」\(p176\)](#)、[「ferror」\(p177\)](#)、[「fopen」\(p184\)](#)、[「fseek」\(p201\)](#)、[「rewind」\(p222\)](#)

例 25.1 clearerr() の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;

    static char name[] = "myfoo";
    char buf[80];

    // 出力ファイルの生成
    if ( (f = fopen(name, "w")) == NULL ) {
        printf("Can't open %s.\n", name);
        exit(1);
    }
    // テキストをファイルに出力
```

```
fprintf(f, "chair table chest\n");
fprintf(f, "desk raccoon\n");

// close the file
fclose(f);

// 再び入力として同じファイルをオープン
if ( (f = fopen(name, "r")) == NULL) {
    printf("Can't open %s.\n", name);
    exit(1);
}

// ファイルの終わりまで全てのテキストファイルを読み込む
for (; feof(f) == 0; fgets(buf, 80, f))
    fputs(buf, stdout);

printf("feof() for file %s is %d.\n", name, feof(f));
printf("Clearing end-of-file status. . .\n");
clearerr(f);
printf("feof() for file %s is %d.\n", name, feof(f));

// ファイルをクローズする
fclose(f);

return 0;
}
```

Output

```
chair table chest
desk raccoon
feof() for file myfoo is 256.
Clearing end-of-file status. . .
feof() for file myfoo is 0.
```

fclose

オープンされたファイルをクローズします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
-------------	-------------	-----------------	---------------	----------------	--------------	--

プロトタイプ #include <stdio.h>
int fclose(FILE *stream);

引数 この関数の引数です。

stream FILE * FILE ストリームへのポインタ

注意点 `fclose()` 関数は、`fopen()`、`freopen()`、`tmpfile()` で生成されたファイルをクローズします。関数は任意のバッファリングされたデータをファイルにフラッシュし、ストリームをクローズします。`fclose()` を呼んだ後、`fopen()`、`freopen()`、`tmpfile()` を用いて再び割り当てなければ、ストリームはもはや無効であり、ファイル関連の関数を用いることはできません。

プログラム中のオープンされた全てのストリームは、プログラムが正常終了すると、フラッシュされクローズされます。

`fclose()` は、`tmpfile()` で生成されたファイルを同時に削除します。

注意： `embedded/RTOS` システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイルにのみ実装されています。

戻り値 `fclose()` は成功すると 0、失敗すると -1 を返します。

参照 [「fopen」\(p184\)](#)、[「freopen」\(p196\)](#)、[「tmpfile」\(p233\)](#)、[「abort」\(p244\)](#)、[「exit」\(p256\)](#)

例 25.2 `fclose()` の使用例

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    FILE *f;
    static char name[] = "myfoo";

    // 出力ファイルの生成
    if ( (f = fopen(name, "w")) == NULL) {
        printf("Can't open %s.\n", name);
        exit(1);
    }
    // ファイルにテキストを出力
    fprintf(f, "pizza sushi falafel\n");
    fprintf(f, "escargot sprocket\n");

    // ファイルをクローズ
    if (fclose(f) == -1) {
        printf("Can't close %s.\n", name);
        exit(1);
    }
}
```

```
    return 0;
}
```

fdopen

ファイル記述子をストリームに変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`FILE *fdopen(int fildes, char *mode);`

注意： Mac OS での `fdopen()` は、`unix.h` ヘッダの中で定義されています。

引数 この関数の引数です。

fildes	int	ファイルの説明
mode	char *	ファイルのオープンモード

注意点 この関数はファイル記述子 `fildes` に対するストリームを生成します。`fprintf()` や `getchar()` のような標準の I/O 関数を使ってこのストリームを利用することができます。CodeWarrior C/C++ では、`mode` 引数の値は無視されます。

戻り値 `fdopen()` は成功するとストリームを返し、エラーに遭遇すると `NULL` を返します。

参照 [「fileno」\(p342\)](#)、[「open」\(p56\)](#)

例 25.3 fdopen() の使用例

```
#include <stdio.h>
#include <unix.h>

int main(void)
{
    int fd;
    FILE *str;

    fd = open("mytest", O_WRONLY | O_CREAT);

    /* ファイル記述子へ書き込み */
    write(fd, "Hello world!\n", 13);
    /* ファイル記述子をストリームに変換 a stream */
```

```
str = fdopen(fd, "w");

/* ストリームへ書き込み the stream */
fprintf(str, "My name is %s.\n", getlogin());

/* ストリームのクローズ */
fclose(str);
/* ファイル記述子のクローズ file descriptor */
close(fd);

return 0;
}
```

feof

ストリームのファイルの終わりの状態を確認します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int feof(FILE *stream);`

引数 この関数の引数です。

stream FILE * FILE ストリームへのポインタ

注意点 `feof()` 関数は、ストリーム上で最後の読み込み操作でのファイルの終わりの状態を調べます。この関数はファイルの終わりの状態をリセットしません。

注意： embedded/RTOS システム上では、この関数は `stdin`、`stdout`、`stderr` ファイルにのみ実装されています。

戻り値 `feof()` は、ストリームがファイルの終わりである場合、非ゼロの値を返し、ファイルの終わりではない場合、ゼロを返します。

参照 [「clearerr」\(p172\)](#)、[「ferror」\(p177\)](#)

例 25.4 feof() の使用例

```
#include <stdio.h>
#include <stdlib.h>
```



```
int main(void)
{
    FILE *f;
    static char filename[80], buf[80] = "";

    // ユーザからのファイル名の設定
    printf("Enter a filename to read.\n");
    gets(filename);

    // 入力用ファイルのオープン
    if (( f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // feof() がファイルの終わりを指さない場合、
    // ファイルからテキスト行を読み込む
    for (; feof(f) == 0 ; fgets(buf, 80, f) )
        printf(buf);

    // ファイルをクローズする
    fclose(f);

    return 0;
}
```

Output:

```
Enter a filename to read.
itwerks
The quick brown fox
jumped over the moon.
```

ferror

ストリームのエラーの状態を確認します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
int ferror (FILE *stream);
```

引数 この関数の引数です。

stream FILE * FILE ストリームへのポインタ

注意点 `ferror()` 関数は、ストリーム上の最後の読み書き操作でのエラー状態を返します。関数は、エラー状態をリセットしません。

注意： `embedded/RTOS` システム上では、この関数は `stdin`、`stdout`、`stderr` ファイルにのみ実装されています。

戻り値 `ferror()` は、ストリームのエラー状態がオンになっていれば非ゼロを返し、エラー状態がオフになっていればゼロを返します。

参照 [「clearerr」\(p172\)](#)、[「feof」\(p176\)](#)

例 25.5 `ferror()` の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    char filename[80], buf[80];
    int ln = 0;

    // ユーザからのファイル名を得る
    printf("Enter a filename to read.\n");
    gets(filename);

    // 入力用ファイルをオープンする
    if ((f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // ファイルの終わりまで一度に 1 行ずつファイルを読む
    do {
        fgets(buf, 80, f);
        printf("Status for line %d: %d.\n", ln++, ferror(f));
    } while (feof(f) == 0);

    // ファイルをクローズする
    fclose(f);

    return 0;
}
```

Output:
Enter a filename to read.

```
itwerks
Status for line 0: 0.
Status for line 1: 0.
Status for line 2: 0.
```

fflush

ホスト環境に対するストリームバッファを空にします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int fflush(FILE *stream);`

引数 この関数の引数です。

stream FILE * FILE ストリームへのポインタ

注意点 `fflush()` 関数は、ストリームに対応したファイルへのバッファを空にします。直前の操作が入力でない場合にストリームが出力ストリームや更新ストリームを指していると、`fflush` 関数はそのストリームで書き込まれていないデータをホスト環境へ送ってファイルへ書き込みます。それ以外の場合の挙動は定義されていません。

入力操作の後では `fflush()` 関数は使わないでください。

注意： `embedded/RTOS` システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

戻り値 `fflush()` は失敗すると非ゼロを返し、成功するとゼロを返します。

参照 [「setvbuf」\(p229\)](#)

例 25.6 fflush() の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    int count;
```

```
// 出力用の新しいファイルを生成
if (( f = fopen("foofoo", "w")) == NULL) {
    printf("Can't open file.\n");
    exit(1);
}
for (count = 0; count < 100; count++) {
    fprintf(f, "%5d\n", count);
    if (count % 10)
        fflush(f); // 10 回毎にバッファのフラッシュ
}
fclose(f);

return 0;
}
```

fgetc

ストリームから次の文字を読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int fgetc(FILE *stream);`

引数 この関数の引数です。

stream FILE * FILE ストリームへのポインタ

注意点 `fgetc()` 関数は、ストリームから次の文字を読み込み、ファイル位置指標を進めます。

注意： embedded/RTOS システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

戻り値 `fgetc()` は、文字を整数として返します。ファイルの終わりに到達した場合、`fgetc()` は EOF を返します。

ファイルを更新モード (+) でオープンしている場合に最後の読み取りまたは書き込みがファイルの終点へ達していなければ、ファイルの読み取りはできず、また位置関数 (`fseek()`、`fsetpos()`、`rewind()`) のいずれかを使ってファイル位置を変更しなければファイルへの書き込みもできません。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「getc」\(p206\)](#)、[「getchar」\(p208\)](#)

例 25.7 fgetc() の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    char filename[80], c;

    // ユーザからファイル名を取得する
    printf("Enter a filename to read.\n");
    gets(filename);

    // 入力用ファイルをオープンする
    if (( f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // ファイルの終わりまで、
    // 一度に文字ファイルから読み込む
    while ( (c = fgetc(f)) != EOF)
        putchar(c); // 文字のプリント

    // ファイルをクローズする
    fclose(f);

    return 0;
}
```

fgetpos

ストリームの現在のファイル位置指標の値を取得します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
int fgetpos(FILE *stream, fpos_t *pos);
```

引数 この関数の引数です。

stream	FILE *	FILE ストリームへのポインタ
pos	fpos_t	ファイル位置タイプへのポインタ

注意点 `fgetpos()` 関数は、`fsetpos()` と組み合わせて、ファイルのランダムアクセスを行います。`fgetpos()` は、コンソール (`stdin`、`stdout`、`stderr`) に関係するストリームで用いた場合、結果に信頼性がありません。

`fseek()` と `ftell()` 関数は、ファイル位置指標を読み込んだり、設定するために `long integer` を利用しますが、`fgetpos()` や `fsetpos()` はより大きいファイルを操作するために `fpos_t` の値を利用します。`fpos_t` 型は、`stdio.h` で定義され、`long int` 型ではないファイル位置指標を持つことができます。

`fgetpos()` は、ストリームに関するファイル位置指標の現在の値を `fpos_t` 型の変数 `pos` が指し示すところに記憶します。

注意： `embedded/RTOS` システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

戻り値 `fgetpos()` は、成功すればゼロを返し、失敗すれば非ゼロを返します。

参照 [「fseek」\(p201\)](#)、[「fsetpos」\(p204\)](#)、[「ftell」\(p204\)](#)

例 25.8 `fgetpos()` の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    fpos_t pos;
    char filename[80], buf[80];

    // ユーザからファイル名を取得する
    printf("Enter a filename to read.\n");
    gets(filename);

    // 入力用ファイルをオープンする
    if ((f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }
    printf("Reading each line twice.\n");

    // 最初のファイル位置指標の値を取得する
    // (which is at the beginning of the file)
    fgetpos(f, &pos);

    // ファイルの終わりに到達するまでそれぞれの行を読み込む
```

```
while (fgets(buf, 80, f) != NULL) {
    printf("Once: %s", buf);

    // 再び読み込むために行の始めに移動する
    fsetpos(f, &pos);
    fgets(buf, 80, f);
    printf("Twice: %s", buf);

    // 次の行のファイル位置を取得する
    fgetpos(f, &pos);
}

// ファイルをクローズする
fclose(f);

return 0;
}
```

Output:
Enter a filename to read.
myfoo
Reading each line twice.
Once: chair table chest
Twice: chair table chest
Once: desk raccoon
Twice: desk raccoon*/

fgets

ストリームから文字配列を読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <stdio.h>
char *fgets(char *s, int n, FILE *stream);

引数 この関数の引数です。

s	char *	デスティネーション文字列
n	int	読み取る最大 char
stream	FILE *	FILE ストリームへのポインタ

注意点 fgets() 関数は、現在のファイル位置で始まるストリームから連続的に文字を読み込み、文字配列として組み立てます。この関数は、n 個の文字が読み込まれたとき、文字の読み込

みを停止します。そして、改行文字 ('\\n') あるいはファイルの終わりに到達すると、読み込みを終了します。

ファイルを更新モード (+) でオープンしている場合に最後の読み取りまたは書き込みがファイルの終点へ達していなければ、ファイルの読み取りはできず、また位置関数 (fseek(), fsetpos(), rewind()) のいずれかを使ってファイル位置を変更しなければファイルへの書き込みもできません。

gets() 関数と違って、fgets() は改行文字 ('\\n') を引数の s に付け加えます。文字配列はヌルで終了します。

注意： embedded/RTOS システム上ではこの関数は stdin、stdout、stderr ファイル用にのみ実装されています。

戻り値 fgets() は成功すると、s へのポインタを返します。文字を読む前にファイルの終わりに到達すると、s は何も変更されず、fgets() はヌルポインタ (NULL) を返します。エラーが発生した場合、fgets() はヌルポインタを返し、s の内容は破損している可能性があります。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「gets」\(p209\)](#)

例 25.9 fgets() の使用例

feof() に関する [「feof\(\) の使用例」\(p176\)](#) を参照してください。

fopen

ストリームとしてファイルをオープンします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
FILE *fopen(const char *filename,
            const char *mode);
```

引数 この関数の引数です。

filename	const char *	開くファイルの名前
mode	const char *	ファイルオープンモード

注意点 `fopen()` 関数は、`filename` で指定されたファイルをオープンし、ストリームを対応させます。`fopen()` 関数は、`FILE` へのポインタを返します。I/O の操作実行時、このポインタはファイルを参照するために利用されます。

モード引数でファイルの使い方を指定します。「[fopen\(\) のオープンモード](#)」でモードの値について説明します。

更新モード：更新モード（+）でオープンしたファイルはバッファ化されます。書き込み操作と読み取り操作が、ストリームバッファをフラッシュする操作で区切られていなければ、ファイルの読み取りはできず、またファイルへの書き込みもできません。これは `fflush()` 関数、またはファイル位置操作（`fseek()`、`fsetpos()`、`rewind()`）のいずれかで可能です。同様に、直前の読み取りまたは書き込みがファイルの終点に達していない限り、ファイル位置関数のいずれかを使ってファイル位置を変更しなければファイルの読み取りはできず、またファイルへの書き込みもできません。

追加モード（`a`、`a+`、`ab`、`ab+`）を除く全てのファイルモードでは、ファイル位置指標がファイルの始まりに設定されます。追加モードではファイル位置指標がファイルの終わりに設定されます。

注意： 書き込みモードおよび読み書き（`w+`、`wb+`）モードでも、ファイルがオープンされたときにあらゆるカレントデータを削除します。

表 25.1 `fopen()` のオープンモード

モード	説明
<code>r</code>	読み込み専用として既存のテキストファイルをオープン。
<code>w</code>	書き込み用に新しいテキストファイルを作成。または既存のファイルを開いて破棄。
<code>a</code>	追加のための既存のテキストファイルをオープン。存在しない場合、新しいファイルを生成。書き込みはファイルの終わりの位置から始まる。
<code>r+</code>	更新モード。既存のテキストファイルを読み書きするためにオープン（注意点を参照）。
<code>w+</code>	更新モード。書き込み用に新しいテキストファイルを作成。または読み取り、書き込み用に既存のファイルを開いて破棄（注意点を参照）。
<code>a+</code>	更新モード。書き込んだり、読み込んだりするために、既存のテキストファイルをオープン、または新しいファイルを生成。書き込みはファイルの終わりの位置から始まる。
<code>rb</code>	読み込み専用として既存のバイナリファイルをオープン。
<code>wb</code>	新しいバイナリファイルを生成。または書き込みのために既存のファイルを開いて破棄。

モード	説明
ab	追加のための既存のバイナリファイルのオープン、存在しない場合、新しいファイルを生成。ファイルの終わりから書き込みが始まる。
r+b または rb+	更新モード。読み書き用に既存のバイナリファイルをオープン。(注意点を参照)
w+b または wb+	更新モード。新しいバイナリファイルを生成、読み書き用に既存のバイナリファイルを開いて破棄(注意点を参照)。
a+b または ab+	更新モード。読み書き用に既存のバイナリファイルのオープン、存在しない場合、新しいファイルを生成。書き込みはファイルの終わりの位置から始まる(注意点を参照)。

戻り値 `fopen()` は、指定された操作に対して指定されたファイルのオープンが成功した場合、`FILE` へのポインタを返します。成功しなかった場合、ヌルポインタ (`NULL`) を返します。

参照 [「fclose」\(p173 \)](#)

例 25.10 fopen() の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    int count;

    // 出力用の新しいファイルを生成
    if (( f = fopen("foofoo", "w")) == NULL) {
        printf("Can't create file.\n");
        exit(1);
    }

    // 0 から 9 の数字を出力
    for (count = 0; count < 10; count++)
        fprintf(f, "%5d", count);

    // ファイルをクローズ
    fclose(f);

    // 追加のためのファイルをオープン
    if (( f = fopen("foofoo", "a")) == NULL) {
        printf("Can't append to file.\n");
        exit(1);
    }

    // 10 から 19 の数字を出力
```

```
for (; count < 20; count++)
    fprintf(f, "%5d\n", count);

// ファイルをクローズ
fclose(f);

return 0;
}
```

fprintf

ストリームに書式化されたテキストを送ります。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int fprintf(FILE *stream,`
`const char *format, ...);`

引数 この関数の引数です。

stream	FILE *	FILE ストリームへのポインタ
format	const char *	書式化した文字列

注意点 `fprintf()` 関数は、書式化されたテキストをストリームに書き込み、ファイル位置指標を進めます。この操作は `printf()` と同じで、ストリームの引数が加わっている点が異なります。 `printf()` の記述を参照してください。

ファイルを更新モード (+) でオープンしている場合に書き込み操作と読み取り操作がストリームバッファをフラッシュする操作で区切られていなければ、ファイルの読み取りはできず、またファイルへの書き込みもできません。これは `fflush()` 関数、またはファイル位置操作 (`fseek()`、`fsetpos()`、`rewind()`) のいずれかで可能です。

注意： `embedded/RTOS` システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

出力制御文字列と変換指定子

書式用の文字配列は通常のテキストと変換仕様です。変換仕様は、書式中において同じ順番のマッチング引数を持つ必要があります。

書式文字列のそれぞれの項目は左から右の順番に並ぶと、ANSI の標準規格で指定されています。

1 個のパーセント記号 (%)

オプション：フラグ -, +, 0, # あるいは空白

オプション：最小幅指定

オプション：精度指定 (小数点以下の桁数)

オプション：サイズ指定

変換オペレータ c, d, e, E, f, g, G, i, n, o, p, s, u, x, X または %

変換仕様は、関連した引数が変換されるように書式に記述されます。仕様はパーセント記号 (%) で始まり、必要に応じてオプションのフラグ文字、オプションの最小幅、オプションの精度の幅、そして変換の型で終わります。2 つのパーセント記号 (%%) の場合、1 個の % を出力します。

オプションのフラグ文字は、出力を書式を変更します。左あるいは右揃えが可能、数値ではゼロを埋め込む、または他の書式で出力することが可能です。変換仕様では、複数のフラグ文字を使うことが可能です。[「書式化した出力関数に関する書式修飾子の型」\(p213\)](#) でフラグ文字について説明します。

オプションの最小幅は 1 文字の数字です。変換される値が最小幅より多くの文字がある場合、必要なら拡張されます。最小幅より少ない文字である場合、これはデフォルトですが、右揃えが行われます。'- ' フラグ文字が使われる場合、変換される値は左揃えとなります。

注意： MSL で許可されている最大の「minimum field width」は 509 文字です。

オプションの精度幅は、ピリオド文字 (.) に続く 10 進数の数字の列です。浮動小数点数では、精度幅は 10 進小数点の後に書く数字の数を指定します。整数値では、精度幅は同様に機能し、最小幅指定をキャンセルします。文字配列で利用する場合では、精度幅は出力の最大幅を指定します。

最小幅と精度幅は、10 進数の文字列の代わりにアスタリスク (*) によって指定することもできます。1 つのアスタリスクは、そこにマッチング引数、変換引数、最小幅あるいは精度幅の指定があることを示します。

最後の文字、すなわち変換の型は、変換仕様のマッチング引数に適応した変換を指定します。[「書式化した出力関数に関する書式修飾子の型」\(p213\)](#) では、変換文字について説明します。

Fprintf に対する MSL Altivec 拡張

MSL (Metrowerks Standard Library) は、標準 printf 関数への Altivec 拡張をサポートしています。

% の後と、指定子の前のセパレータ引数には、どんな文字でも使用できます。@ シンボルも使用できます。@ シンボルは非 Motorola 拡張子であり、指定された文字列を指定子として使用します。

'c' 指定子は、任意の char をセパレータとして使うことができますが、'-','+', '#',' ' は使用できません。

「[Altivec Printf 拡張の使用例](#)」(p216) をを参照してください。

表 25.2 書式化した出力関数に関する書式修飾子の型

修飾子	説明
	サイズ
h	h に続く d, i, o, u, x, X 変換指定子は、対応する引数が short int または unsigned short int であることを示す。
l	l に続く d, i, o, u, x, X 変換指定子は、引数が long int または unsigned long int であることを示す。
ll	ll に続く d, i, o, u, x, X 変換指定子は、引数が long long または unsigned long long であることを示す。
L	L に続く e, E, f, g, G 変換指定子は long double を示す。
v	Altivec : c, d, i, o, u, x, X のいずれかが後ろに続く場合、vector bool char、vector signed char、または vector unsigned char。 f が後ろに続く場合、vector float。
vh hv	Altivec : c, d, i, o, u, x, X のいずれかが後ろに続く場合、vector short、vector unsigined short、vector bool short、vector pixel。
vl lv	Altivec : c, d, i, o, u, x, X のいずれかが後ろに続く場合、vector int、vector unsigned int、vector bool int。
	フラグ
-	変換は左に揃える。
+	数値の場合、変換は符号 (+ または -) が前に付加される。デフォルトでは、負の数の値のみマイナス符号 (-) が前に付加される。
space	変換の最初の文字が符号文字でない場合、1 つの空白が前に付加される。空白のフラグはプラスフラグ (+) と組み合わせる場合、プラスの符号フラグは常に符号を数値の前に付加するため効果がない。
#	c, d, i, u の変換の場合、# フラグは効果がない。s の変換の場合、Pascal 文字列のポインタは文字列として出力される。o 変換の場合、# フラグでは、ひとつの 0 が前に付加される。このフラグで x 変換の場合、0x を前に付加する。e, E, f, g, G 変換の場合、# フラグは、出力に 10 進小数点を付け加える。g と G の変換では、削除されない 10 進小数点の後にゼロが並ぶ。

0	このフラグは、変換の左側にゼロを埋め込む。これは、d, i, o, u, x, X, e, E, f, g, G 変換に適応される。通常は空白文字である部分に、符号の後の先頭の 0、基数を示す文字が並ぶ。マイナスの符号フラグは、0 フラグの文字に取って代わる。0 は、d, i, o, u, x, X 変換の型に対して精度幅とともに利用される場合、無視される。
@	Altivec：このフラグは引数によって指定された文字列へのポインタを示す。この文字列はベクトル要素のセパレータをして使用される。
	変換
d	対応する引数は、符号付き 10 進数に変換される。
i	対応する引数は、符号付き 10 進数に変換される。
o	引数は、符号なし 8 進数に変換される。
u	引数は符号なし 10 進数に変換される。
x, X	引数は符号なしの 16 進数に変換される。x 変換の場合は小文字 (abcdef) が使われ、X 変換では大文字 (ABCDEF) が使われる。
n	この変換の型は、これまでの printf() による出力された項目の数を記憶する。これに対応する引数は整数型へのポインタでなければならない。
f	対応する浮動小数点の引数 (float または double) は、10 進表記で出力される。デフォルトの精度は、6 (10 進小数点以下 6 桁)。精度幅が明確に 0 の場合、10 進小数点は出力されない。
e, E	浮動小数点の引数 (float または double) は、指数表現で出力される：[-]b.aaae ± Eee。10 進小数点の前に 1 桁 (b) の数字がある。オプションの精度幅が指定されない場合、デフォルトは 10 進小数点から 6 桁である。精度幅が 0 なら 10 進小数点は出力されない。指数 (ee) は、少なくとも 2 桁である。 e 変換では、指数の前に小文字の e が付加されるのは E の変換は、大文字 E が指数の前に付加される。
g, G	g 変換では、f あるいは e 変換が使われ、G 変換では F や E の変換が使われる。変換の e (または E) は、変換された指数が -4 より小さいかあるいは精度幅より大きいときに限り利用される。精度幅では有効数字を指定する。10 進小数点以下に続く数字がなければ、小数点は出力されない。
c	対応する引数は 1 文字として出力される。
s	対応する引数、文字配列を指すポインタは文字列として出力される。文字列の出力は、ヌル文字に到達した時点で終了する。ヌル文字は出力されない。
p	対応する引数はポインタとして扱われる。引数は x 変換の書式を使って出力される。

	CodeWarrior での拡張
#s	対応する引数、すなわち Pascal 文字列へのポインタは、文字列として出力される。Pascal 文字列はバイト長で指定された数字の文字が連続するバイト長である。 注意：この変換は ANSI C ライブラリに対しての拡張で、他の書式のバリエーションには同じ方法で適応されない。

戻り値 `fprintf()` は書き込まれた引数の数を返し、エラーが発生した場合、負の数を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「printf」\(p211\)](#)、[「sprintf」\(p231\)](#)、[「vfprintf」\(p237\)](#)、[「vprintf」\(p239\)](#)、[「vsprintf」\(p240\)](#)

例 25.11 `fprintf()` の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    static char filename[] = "myfoo";
    int a = 56;
    char c = 'M';
    double x = 483.582;

    // 出力用の新しいファイルを生成
    if (( f = fopen(filename, "w")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // ファイルに書式化されたテキストを出力
    fprintf(f, "%10s %4.4f %-10d\n%10c", filename, x, a, c);

    // ファイルをクローズ
    fclose(f);

    return 0;
}
```

`fputc`

ストリームに文字を書き込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int fputc(int c, FILE *stream);`

引数 この関数の引数です。

c	int	ファイルへ書き込む文字
stream	FILE *	FILE ストリームへのポインタ

注意点 `fputc()` 関数は、文字 `c` をストリームに書き込み、ストリームのファイル位置指標を進ませます。引数 `c` は、`int` 型ですが、ストリームに書き込まれる前に `char` 型に変換されます。`fputc()` は、関数として書かれておりマクロではありません。

ファイルを更新モード(+)でオープンしている場合に書き込み操作と読み取り操作がストリームバッファをフラッシュする操作で区切られていなければ、ファイルの読み取りはできず、またファイルへの書き込みもできません。これは `fflush()` 関数、またはファイル位置操作 (`fseek()`、`fsetpos()`、`rewind()`) のいずれかで可能です。

注意： `embedded/RTOS` システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

戻り値 `fputc()` は成功した場合、書き込まれた文字を返し、失敗した場合、`EOF` を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「putc」\(p217\)](#)、[「putchar」\(p218\)](#)

例 25.12 `fputc()` の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    int letter;

    // 出力用の新しいファイルを生成
    if (( f = fopen("foofoo", "w")) == NULL) {
        printf("Can't create file.\n");
        exit(1);
    }

    // 一度に 1 文字ずつアルファベットを出力
    // at a time
    for (letter = 'A'; letter <= 'Z'; letter++)
```



```
    fputc(letter, f);  
    fclose(f);  
  
    return 0;  
}
```

fputs

ストリームに文字配列を書き込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int fputs(const char *s, FILE *stream);`

引数 この関数の引数です。

s	const char *	ファイルへ書き込む文字列
stream	FILE *	FILE ストリームへのポインタ

注意点 `fputs()` 関数は、引数 `s` によって指し示された配列をストリームに書き込み、ファイル位置指標を進ませます。この関数は `s` の中の全ての文字を書き込みますが、最後のヌル文字は含みません。`puts()` とは違い、`fputs()` は `s` の出力の最後で改行 ('`\n`') を出力しません。

ファイルを更新モード (+) でオープンしている場合に書き込み操作と読み取り操作がストリームバッファをフラッシュする操作で区切られていなければ、ファイルの読み取りはできず、またファイルへの書き込みもできません。これは `fflush()` 関数、またはファイル位置操作 (`fseek()`、`fsetpos()`、`rewind()`) のいずれかで可能です。

注意： `embedded/RTOS` システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

戻り値 `fputs()` は成功した場合、ゼロを返し、失敗した場合、非ゼロを返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「puts」\(p219\)](#)

例 25.13 fputs() の使用例

```
#include <stdio.h>  
#include <stdlib.h>  
  
int main(void)
```

```
{
    FILE *f;

    // 出力用として新しいファイルを生成
    if (( f = fopen("foofoo", "w")) == NULL) {
        printf("Can't create file.\n");
        exit(1);
    }

    // 文字列をファイルに出力
    fputs("undo\n", f);
    fputs("copy\n", f);
    fputs("cut\n", f);
    fputs("rickshaw\n", f);

    // ファイルをクローズ
    fclose(f);

    return 0;
}
```

fread

ストリームからバイナリデータを読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
size_t fread(void *ptr, size_t size,
             size_t nmemb, FILE *stream);
```

引数 この関数の引数です。

ptr	void *	読み取り先へのポインタ
size	size_t	指された配列のサイズ
nmemb	size_t	読み取る要素の数
stream	FILE *	FILE ストリームへのポインタ

注意点 fread() 関数は、バイナリやテキストデータのブロックを読み込みファイル位置指標を更新します。ストリームから読み込まれたデータは、ptr によって指し示される配列に記憶されます。引数の size と nmemb には、読み込む項目のサイズと、項目の数を記述します。

ファイルの終点に達していないまたは読み取りエラーが起きない限り、fread() 関数は reads nmemb 項目を読みとります。

ファイルを更新モード (+) でオープンしている場合に直前の読み取りまたは書き込みがファイルの終点に達していない限り、ファイル位置関数 (fseek()、fsetpos()、rewind()) のいずれかを使ってファイル位置を変更しなければファイルの読み取りはできず、またファイルへの書き込みもできません。

注意: embedded/RTOS システム上ではこの関数は stdin、stdout、stderr ファイル用にのみ実装されています。

戻り値 読み取りが成功した場合、fread() 関数は項目の数を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「fgets」\(p183\)](#)、[「fwrite」\(p205\)](#)

例 25.14 fread() の使用例

```
#include <stdio.h>
#include <stdlib.h>

// 項目のサイズ (バイト数) を定義
#define BUFSIZE 40

int main(void)
{
    FILE *f;
    static char s[BUFSIZE] = "The quick brown fox";
    char target[BUFSIZE];

    // 入出力用の新しいファイルを生成
    if ((f = fopen("foo", "w+")) == NULL) {
        printf("Can't create file.\n");
        exit(1);
    }

    // fwrite() を使ってストリームへ出力
    fwrite(s, sizeof(char), BUFSIZE, f);

    // ファイルの始まりへ移動
    rewind(f);

    // fread() を使ってストリームから読み込む
    fread(target, sizeof(char), BUFSIZE, f);

    // コンソールへ結果を出力
    puts(s);
    puts(target);
}
```

```
// ファイルをクローズ
fclose(f);

return 0;
}
```

```
Output:
The quick brown fox
The quick brown fox
```

freopen

ストリームを他のファイルヘリダイレクトします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

```
プロトタイプ #include <stdio.h>
FILE *freopen(const char *filename,
               const char *mode, FILE *stream);
```

引数 この関数の引数です。

filename	const char *	再オープンするファイルの名前
mode	const char *	ファイルのオープンモード
stream	FILE *	FILE ストリームへのポインタ

注意点 fopen() 関数は、関係するファイルストリームを他のファイルに変更します。関数は、最初、関係するファイルをクローズし、ファイル名 (filename)、指定されたモード (mode) 同じストリームを使って、新たなファイルをオープンします。

戻り値 fopen() は成功した場合、ストリームの値を返します。失敗した場合、ヌルポインタ (NULL) を返します。

参照 [「fopen」\(p184\)](#)

例 25.15 freopen() の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
```

```
// コンソールから新しいファイルへ出力をリダイレクト
if (( f = freopen("newstdout", "w+", stdout)) == NULL) {
    printf("Can't create new stdout file.\n");
    exit(1);
}
printf("If all goes well, this text should be in\n");
printf("a text file, not on the screen via stdout.\n");
fclose(f);

return 0;
}
```

fscanf

ストリームから書式化されたテキストを読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int fscanf(FILE *stream,`
`const char *format, ...);`

引数 この関数の引数です。

stream	FILE *	FILE ストリームへのポインタ
format	const char *	書式化した文字列

注意点 `fscanf()` 関数は、プログラマが定義した書式化されたテキストをストリームより読み込みます。この関数は、`scanf()` 関数とまったく同じように動作し、読み込まれるストリームを指す引数 `stream` が追加されています。 `scanf()` の記述を参照してください。

ファイルを更新モード (+) でオープンしている場合に直前の読み取りまたは書き込みがファイルの終点に達していない限り、ファイル位置関数 (`fseek()`、`fsetpos()`、`rewind()`) のいずれかを使ってファイル位置を変更しなければファイルの読み取りはできず、またファイルへの書き込みもできません。

注意： `embedded/RTOS` システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

入力制御文字列と変換指定子

書式用の引数は通常のテキストと空白文字 (スペース、タブ、改行) 変換仕様を含む文字配列です。通常のテキストは入力ストリームに一致する文字を指定します。空白文字は空

白文字以外の文字に達するまでスキップされます。変換仕様は入力ストリーム内のどの文字を変換して保存するかを指定します。

変換仕様は書式中において同じ順番のマッチング引数を持つ必要があります。scanf() はメモリにデータを保存し、変換仕様のマッチング引数は、関連するオブジェクト型へのポインタになるためです。

変換仕様はパーセント記号(%)で始まり、必要に応じてオプションの最小幅、オプションの代入抑制文字、そして変換の型で終わります。2 つのパーセント記号(%%)の場合、1 個の % を出力します。

オプションの幅は入力フィールドの最大幅を指定する 10 進数です。scanf() は指定された幅以上の文字は読み取りません。

オプションの代入抑制文字(*)は項目を読み取るけれども代入をしないため、項目をスキップするためにも使用できます。代入抑制の変換仕様には対応する引数が必要です。

最後の文字、すなわち変換の型は、必要な変換を指定します。[「書式化した入力関数に関する書式修飾子の型」](#)で変換文字について説明します。

Scanf への MSL Altivec 拡張

MSL は、標準 printf 関数への Altivec 拡張をサポートしています。

% の後と、指定子の前のセパレータ引数には、どんな文字でも使用できます。@ シンボルも使用できます。@ シンボルは非 Motorola 拡張子であり、指定された文字列を指定子として使用します。

'c' 指定子は、任意の char をセパレータとして使うことができますが、'-','+', '#',' ' は使用できません。

[「Altivec Scanf 拡張の使用例」\(p227\)](#) をを参照してください。

表 25.3 書式化した入力関数に関する書式修飾子の型

修飾子	説明
	サイズ修飾子
h	h は対応する引数が short int または unsigned short int であることを示す。
l	整数変換修飾子と一緒に使う場合、l は long int または unsigned long int であることを示す。浮動小数点変換修飾子と一緒に使う場合、l は double であることを示す。
ll	整数変換修飾子と一緒に使う場合、ll は引数が long long または unsigned long long であることを示す。
L	L は対応する浮動小数点変換修飾子が long double を示す。

v	Altivec : c、d、i、o、u、x、X のいずれかが後ろに続く場合、vector bool char、vector signed char、または vector unsigned char。 f が後ろに続く場合、vector float。
vh hv	Altivec : c、d、i、o、u、x、X のいずれかが後ろに続く場合、vector short、vector unsinged short、vector bool short、vector pixel。
vl lv	Altivec : c、d、i、o、u、x、X のいずれかが後ろに続く場合、vector long、vector unsigned long、vector bool。
変換修飾子	
d	10 進の整数を読み取る。
i	10 進、8 進、16 進の整数を読み取る。整数にはプラスまたはマイナス符号 (+, -) 8 進数には 0、16 進数には 0x または 0X が頭に付加される。
o	8 進の整数を読み取る。
u	符号なし 10 進の整数を読み取る。
x, X	16 進の整数を読み取る。
e, E, f, g, G	浮動小数点の数値を読み取る。10 進 (3456.483) または指数表記 ([-] b . aae ± dd) も可。
s	文字列をを読み取る。入力文字列は空白文字、または最大幅に達したとき終了したと見なされる。配列の最後にヌル文字が追加される。
c	1 文字を読み取る。空白文字をスキップせず、この変換型で読み取る。
p	ポインタアドレスを読み取る。printf() による p 変換では、入力形式は出力形式と同じになる。
n	この変換は入力ストリームからは読みとらないが、読みとった文字数を対応する引数に保存する。
[scanset]	入力ストリームキャラクタを読み取り、scanset で指定されたフィルタを事項する。 「Scanset」(p199) を参照。

Scanset

変換指定子 %[で、scanset を指定します。scanset とは連続した文字列で、scanset に対応する引数によって読み取られ、文字列へ格納されます。[と] の間にあるキャラクタが scanset を定義します。末尾にヌル文字が付加されます。

入力ストリームキャラクタは、scanset 以外のキャラクタが見つかるまで読み取られます。scanset の第一キャラクタが曲折アクセント符号 (^) である場合、scanset のキャラクタが読み取られるまで、入力ストリームキャラクタは読み取られます。ヌル文字がキャラクタ配列の末尾に付加されます。

変換指定子が `%[abcdef]` ならば、`scanset` は `abcdef` です。a から f までのキャラクタは受け付けられ、格納されます。このセット以外のキャラクタに遭遇したとき、読み取りと格納は停止します。

```
char str[20];
```

例えば上記のように定義した場合、

```
sscanf("acdfxbe", "%[abcdef]", str);
```

を実行すると、`acdf` を `str` へ格納します。x とそれ以降のキャラクタは格納されません。x は `scanset` ではないからです。

`scanset` の第一キャラクタが `^` である場合、それ以降のキャラクタは読み取りと格納を停止させるキャラクタセットを定義します。このように定義された `scanset` 以外のキャラクタは受け付けられます。これを `exclusionary scanset` と呼びます。

```
sscanf("stuvawxyz", "%[^abcdef]", str);
```

を実行すると、`stuv` を `str` へ格納します。`^` を `scanset` に含めるには、第一キャラクタを `^` 以外のキャラクタにします。`^` が第一キャラクタの場合、`exclusionary scanset` のメンバーを示すものとして解釈されます。`%[^abc]` は `abc` を `exclusionary scanset` として定義します。`%[a^bc]` は `abc^` を `scanset` として定義します。`%[^a^bc]` は `exclusionary scanset` として `abc^` を定義します。

`]` を `scanset` に含めるには、`%[` の直後に `]` を置き、`scanset` の第一キャラクタにします(例: `%[]abc c`)。 `exclusionary scanset` に含めるには、`^` の直後に置きます(例: `%[^]ab`)。これ以外の場所にある `]` は、`scanset` の定義を終えるものとして解釈されます。

- を `scanset` に含めるには、リストの先頭が最後、または最初の `^` の後に置きます(例: `%[-abc]`、`%[abc-]`、`%[^-abc]`、`%[^abc-]`)。C 規格では以下のように記述されています。

- キャラクタが `scanlist` にあり、かつ第一キャラクタではなく、第一キャラクタ `^` の次でも、最後のキャラクタでもない場合、挙動は実装によって定義されます。

MSL は `scanlist` にある - を、キャラクタの範囲を定義するものとして解釈します。つまり、`%[a-z]` という定義は、`%[abcdefghijklmnopqrstuvwxyz]` と等価です。これは MSL の実装であり、その他の C ライブラリによる解釈は異なる場合があります。また - の前にある数値は、その後に続く数値よりも少ないと解釈されますので注意してください。もしこの関係が未定義のままでなければ、予期せぬ結果を生じます。

戻り値 `fscanf()` は項目の数を返します。データ読み込み中に、書式の文字列と矛盾するようなエラーがある場合、`fscanf()` は `errno` を非ゼロの値に設定します。ファイルの終わりに到達した場合、`fscanf()` は EOF を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「errno」\(p49\)](#)、[「scanf」\(p224\)](#)

例 25.16 fscanf() の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    int i;
    double x;
    char c;

    // 入出力用として新しいファイルを生成
    if ((f = fopen("foobar", "w+")) == NULL) {
        printf("Can't create new file.\n");
        exit(1);
    }

    // ファイルに書式化されたテキストを出力
    fprintf(f, "%d\n%f\n%c\n", 45, 983.3923, 'M');

    // ファイルの始まりへ移動
    rewind(f);

    // fscanf() を使ってストリームから読み込み
    fscanf(f, "%d %lf %c", &i, &x, &c);

    // ファイルをクローズ
    fclose(f);

    printf("The integer read is %d.\n", i);
    printf("The floating point value is %f.\n", x);
    printf("The character is %c.\n", c);

    return 0;
}
```

Output:
The integer read is 45.
The floating point value is 983.392300.
The character is M.

fseek

ファイル位置指標を移動させます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int fseek(FILE *stream, long offset, int whence);`

引数 この関数の引数です。

stream	FILE *	FILE ストリームへのポインタ
offset	long	移動するオフセットのバイト数
whence	int	オフセットの開始位置

注意点 `fseek()` 関数は、ファイルヘランダムアクセスを可能にするために、ファイル位置指標を移動させます。

この関数は絶対的あるいは相対的に位置指標を動かします。引数 `whence` は、`stdio.h` の中で定義された 3 つの値、`SEEK_SET`、`SEEK_CUR`、`SEEK_END` のうちどれか 1 つを持つことができます。

`SEEK_SET` の値の場合、ファイルの始まりから引数 `offset` の内容をバイト数として、ファイル位置指標に反映されます。この場合、`offset` はゼロ以上でなければなりません。

`SEEK_CUR` の値の場合、現在の位置からの `offset` バイトがファイル位置指標に反映されます。引数 `offset` は負あるいは正の値でよい。

`SEEK_END` は、ファイルの終わりから `offset` バイトがファイル位置指標に反映されます。引数 `offset` はゼロ以下である必要があります。

`fseek()` 関数は直前の `ungetc()` 呼び出しを取り消して、`stream` の EOF 状態をクリアします。

注意： 以下はテキストモードで開いた MS DOS テキストファイルで可能な `fseek` 関数には制限があります。キャリッジリターン、ラインフィードの翻訳に問題があるためです。また EOF 翻訳のためにファイルの終末近くでは `fseek` 操作が狂うことがあります。

以下はテキストモードで開いた MS DOS テキストファイルで可能な `fseek` 操作です。

```
ftell() が返したオフセットを使ってファイルの先頭からシークする。  
SEEK_SET、SEEK_CUR と SEEK_END からのゼロオフセットを使ってシークする。
```

注意： `embedded/RTOS` システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

戻り値 `fseek()` は成功した場合はゼロを返し、失敗した場合は非ゼロを返します

参照 [「fgetpos」\(p181\)](#)、[「fsetpos」\(p204\)](#)、[「ftell」\(p204\)](#)

例 25.17 fseek() の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    long int pos1, pos2, newpos;
    char filename[80], buf[80];

    // ユーザからファイル名を取得
    printf("Enter a filename to read.\n");
    gets(filename);

    // 入力用のファイルをオープン
    if ((f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    printf("Reading last half of first line.\n");

    // 最初の行を読み込んだ後、
    // ファイル位置指標を取得
    pos1 = ftell(f);
    fgets(buf, 80, f);
    pos2 = ftell(f);
    printf("Whole line: %s\n", buf);

    // 行の中央を計算
    newpos = (pos2 - pos1) / 2;

    fseek(f, newpos, SEEK_SET);
    fgets(buf, 80, f);
    printf("Last half: %s\n", buf);

    // ファイルをクローズ
    fclose(f);

    return 0;
}
```

Output:
Enter a filename to read.
itwerks
Reading last half of first line.

```
Whole line: The quick brown fox  
Last half: brown fox
```

fsetpos

ファイル位置指標を設定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int fsetpos(FILE *stream, const fpos_t *pos);`

引数 この関数の引数です。

stream	FILE *	FILE ストリームへのポインタ
pos	fpos_t	ファイル位置のタイプへのポインタ

注意点 `fsetpos()` 関数は、`pos` で指し示めされた値を使ってストリームのファイル位置指標を設定します。`fseek()` で使われる `long int` 型で表現される値よりも、大きなサイズを持つファイルを扱う場合、この関数は `fgetpos()` と組み合わせて利用されます。

`fsetpos()` は、`ungetc()` に対する過去の呼び出しを元に戻し、ファイルの終わりの状態をクリアします。

注意： `embedded/RTOS` システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

戻り値 `fsetpos()` は成功するとゼロ、失敗すると非ゼロを返します。

参照 [「fgetpos」\(p181\)](#)、[「fseek」\(p201\)](#)、[「ftell」\(p204\)](#)

例 25.18 `fsetpos()` の使用例

[「fgetpos\(\) の使用例」\(p182\)](#) を参照してください。

ftell

現在のファイル位置指標の値を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`long int ftell(FILE *stream);`

引数 この関数の引数です。

stream FILE * FILE ストリームへのポインタ

注意点 `ftell()` 関数は、ストリームのファイル位置指標の現在の値を返します。これは、`fseek()` と組み合わせて、ファイルに対してのラインダムアクセスを提供する場合に利用されます。

与えられたストリームが `stdin`、`stdout`、`stderr` のコンソールファイルに関連するストリームの場合、正確に動作しないでしょう。それは、ファイル指標の位置が適切ではないからです。`ftell()` は、`long int` 型で表現された値よりも大きなサイズのファイルを扱うことができません。このような場合、`fgetpos()` と `fsetpos()` 関数を利用してください。

注意： `embedded/RTOS` システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

戻り値 `ftell()` は成功した場合、現在のファイル位置指標の値を返します。失敗した場合、`-1L` を返し大域変数の `errno` に非ゼロの値を設定します。

参照 [「errno」\(p49\)](#)、[「fgetpos」\(p181\)](#)

例 25.19 `ftell()` の使用例

[「fseek\(\) の使用例」\(p203\)](#) を参照してください。

fwrite

ストリームにバイナリデータを書き込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`size_t fwrite(const void *ptr, size_t size,`
`size_t nmemb, FILE *stream);`

引数 この関数の引数です。

ptr	void *	書き込まれている項目へのポインタ
size	size_t	書き込まれている項目のサイズ
nmemb	size_t	書き込まれている要素の数
stream	FILE *	FILE ストリームへのポインタ

注意点 fwrite() 関数は、size バイトの nmemb 個の項目をそれぞれストリームに書き込みます。各項目は ptr で指し示された配列に格納します。配列をストリームに書き込んだ後、それに応じてファイル位置指標を進ませます。

ファイルを更新モード(+)でオープンしている場合に書き込み操作と読み取り操作がストリームバッファをフラッシュする操作で区切られていなければ、ファイルの読み取りはできず、またファイルへの書き込みもできません。これは fflush() 関数、またはファイル位置操作 (fseek()、fsetpos()、rewind()) のいずれかで可能です。

注意： embedded/RTOS システム上ではこの関数は stdin、stdout、stderr ファイル用にのみ実装されています。

戻り値 fwrite() は、ストリームに書き込むことが成功した項目の数を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「fread」\(p194\)](#)

例 25.20 fwrite() の使用例

[「fread\(\) の使用例」\(p195\)](#) を参照してください。

getc

ストリームから次の文字を読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
int getc(FILE *stream);
```

引数 この関数の引数です。

stream	FILE *	FILE ストリームへのポインタ
--------	--------	------------------

注意点 `getc()` 関数は、ストリームから次の文字を読み込みファイル位置指標を進ませ、`int` 型の値として文字を返します。`fgetc()` 関数と異なり、`getc()` 関数はマクロとして実装されています。

ファイルを更新モード (+) でオープンしている場合に最後の読み取りまたは書き込みがファイルの終点へ達していなければ、ファイルの読み取りはできず、また位置関数 (`fseek()`、`fsetpos()`、`rewind()`) のいずれかを使ってファイル位置を変更しなければファイルへの書き込みもできません。

戻り値 `getc()` はストリームから次の文字を返します。ファイルの終わりに達しているとき、またはエラーが起きたときに `EOF` を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「fgetc」\(p180\)](#)、[「fputc」\(p191\)](#)、[「getchar」\(p208\)](#)、[「putchar」\(p218\)](#)

例 25.21 `getc()` の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    char filename[80], c;

    // ユーザからファイル名を得る
    printf("Enter a filename to read.\n");
    scanf("%s", filename);

    // 入力用のファイルをオープン
    if ((f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // ファイルの終わりまで1度に1文字を読み込む
    while ( (c = getc(f)) != EOF)
        putchar(c);

    // close the file
    fclose(f);

    return 0;
}
```

getchar

stdin から次の文字を読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int getchar(void);`

引数 なし。

注意点 `getchar()` 関数は、stdin ストリームから 1 文字読み込みます。

注意： `Getchar()` は `getc(stdin)` として実装されています。stdin がバッファされた場合、このような `getchar` の戻り値は遅れるか、最適化されます。ほとんどの場合、stdin はラインバッファ化されて実装されています。

戻り値 `getchar()` は、成功した場合、int 型として stdin から次の文字の値を返します。ファイルの終わりやエラーが発生した場合、EOF を返します。

参照： [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「fgetc」\(p180\)](#)、[「getc」\(p206\)](#)、[「putchar」\(p218\)](#)

例 25.22 getchar() の使用例

```
#include <stdio.h>

int main(void)
{
    int c;

    printf("Enter characters to echo, * to quit.\n");

    // コンソールから * の文字が読み込まれるまで
    // 入力された文字がエコーされる
    while ( (c = getchar()) != '*' )
        putchar(c);

    printf("\nDone!\n");

    return 0;
}
```



```
Output:
Enter characters to echo, * to quit.
I'm experiencing deja-vu *
I'm experiencing deja-vu
Done!
```

gets

stdin から文字配列を読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`char *gets(char *s);`

引数 この関数の引数です。

s char s 書き込まれている文字列

注意点 gets() 関数は、stdin から文字を読み込み、引数 s によって指し示された文字配列に続けて記憶します。文字は、改行あるいはファイルの終わりのいずれかに到達するまで読み込まれます。

fgets() と異なり、プログラマは、読み込む文字の数の限界を指定できません。また、gets() は改行文字 ('\n') を読み込んだ場合、無視するため、ファイル位置指標は次の行へ進み、改行文字は s に記憶されません。fgets()、gets() のようにヌル文字が文字列の終わりとなります。

任意の文字が読まれる前にファイルの終わりに到達した場合、gets() は文字配列 s に影響を与えることなく、ヌルポインタ (NULL) を返します。読み込みエラーが発生した場合、配列の中身は破損している可能性があります。

戻り値 成功した場合は s を返し、失敗した場合はヌルポインタを返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「fgets」\(p183\)](#)

例 25.23 gets() の使用例

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char buf[100];
```

```
printf("Enter text lines to echo.\n");
printf("Enter an empty line to quit.\n");

// 空の行が読み込まれるまで
// コンソールから文字列を読み込む
while (strlen(gets(buf)) > 0)
    puts(buf); // puts() は、その出力に改行を付加する

printf("Done!\n");

return 0;
}
```

Output:

```
Enter text lines to echo.
Enter an empty line to quit.
I'm experiencing deja-vu
I'm experiencing deja-vu
Now go to work
Now go to work

Done!
```

perror

stderr へエラーメッセージを出力します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`void perror(const char *s);`

引数 この関数の引数です。

`s` `const char *` `errno` とメッセージを印刷する

注意点 `perror()` 関数は、引数 `s` によって指し示された文字配列と大域変数の `errno` の値を出力します。

参照 [「abort」\(p244\)](#)、[「errno」\(p49\)](#)

例 25.24 error() の使用例

```
#include <stdio.h>

#define MAXLIST 10

int main(void)
{
    int i[MAXLIST], count;

    printf("Enter %d numbers.\n", MAXLIST);
    printf("Numbers less than 0 will generate an error.\n");

    // コンソールから MAXLIST の整数値を読み込む
    for (count = 0; count < MAXLIST; count++) {
        scanf("%d", &i[count]);

        // その値が 0 以下の場合、
        // エラーメッセージを perror() を使って出力
        if (i[count] < 0)
            perror("Invalid entry!\n");
    }
    printf("Done!\n");

    return 0;
}
```

printf

書式化されたテキストを出力します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
int printf(const char *format, ...);
```

引数 この関数の引数です。

format const char * 書式化した文字列

注意点 printf() 関数は書式化されたテキストを出力します。関数は 1 つ以上の引数を持ち、最初の引数は書式すなわち文字配列のポインタです。書式に続くオプションの引数は、整数値、文字、浮動小数点の値などの項目で、文字列に変換され、指定されたところにその書式の出力が挿入されます。

printf() 関数は stdout にその出力を送ります。

Printf 制御文字列と変換指定子

書式用の文字配列は、通常のテキストと変換仕様です。変換仕様は、書式中において同じ順番のマッチング引数を持つ必要があります。

書式文字列のそれぞれの項目は左から右の順番に並ぶと、ANSI の標準規格で指定されています。

1 個のパーセント記号 (%)

オプション：フラグ -, +, 0, # あるいは空白

オプション：最小幅指定

オプション：精度指定（小数点以下の桁数）

オプション：サイズ指定

変換オペレータ c, d, e, E, f, g, G, i, n, o, p, s, u, x, X または %

変換仕様は、関連した引数が変換されるように書式に記述されます。仕様はパーセント記号 (%) で始まり、必要に応じてオプションのフラグ文字、オプションの最小幅、オプションの精度の幅、そして変換の型で終わります。2 つのパーセント記号 (%%) の場合、1 個の % を出力します。

オプションのフラグ文字は、出力を書式を変更します。左あるいは右揃えが可能、数値ではゼロを埋め込む、または他の書式で出力することが可能です。変換仕様では、1 つ以上のフラグ文字を使うことが可能です。[「書式化した出力関数に関する書式修飾子の型」\(p213\)](#) でフラグ文字について説明します。

オプションの最小幅は 1 文字の数字です。変換される値が最小幅より多くの文字がある場合、必要なら拡張されます。最小幅より少ない文字である場合、これはデフォルトですが、右揃えが行われます。'- ' フラグ文字が使われる場合、変換される値は左揃えとなります。

注意： MSL で許可されている最大の「minimum field width」は 509 文字です。

オプションの精度幅は、ピリオド文字 (.) に続く 10 進数の数字の列です。浮動小数点数では、精度幅は 10 進小数点の後に書く数字の数を指定します。整数値では、精度幅は同様に機能し、最小幅指定をキャンセルします。文字配列で利用する場合では、精度幅は出力の最大幅を指定します。

最小幅と精度幅は、10 進数の文字列の代わりにアスタリスク (*) によって指定することもできます。1 つのアスタリスクは、そこにマッチング引数、変換引数、最小幅あるいは精度幅の指定があることを示します。

最後の文字、すなわち変換の型は、変換仕様のマッチング引数に適応した変換を指定します。[「書式化した出力関数に関する書式修飾子の型」\(p213\)](#) では、変換文字について説明します。

Printf に対する MSL Altivec 拡張

MSL は、標準 printf 関数への Altivec 拡張をサポートしています。

% の後と、指定子の前のセパレータ引数には、どんな文字でも使用できます。@ シンボルも使用できます。@ シンボルは非 Motorola 拡張子であり、指定された文字列を指定子として使用します。

'c' 指定子は、任意の char をセパレータとして使うことができますが、'-','+','#',' ' は使用できません。

[「Altivec Printf 拡張の使用例」\(p216\)](#) を参照してください。

表 25.4 書式化した出力関数に関する書式修飾子の型

修飾子	説明
	サイズ
h	h に続く d, i, o, u, x, X 変換指定子は、対応する引数が short int または unsigned short int であることを示す。
l	l に続く d, i, o, u, x, X 変換指定子は、引数が long int または unsigned long int であることを示す。
ll	ll に続く d, i, o, u, x, X 変換指定子は、引数が long long または unsigned long long であることを示す。
L	L に続く e, E, f, g, G 変換指定子は long double を示す。
v	Altivec : c、d、i、o、u、x、X のいずれかが後ろに続く場合、vector bool char、vector signed char、または vector unsigned char。 f が後ろに続く場合、vector float。
vh hv	Altivec : c、d、i、o、u、x、X のいずれかが後ろに続く場合、vector short、vector unsinged short、vector bool short、vector pixel。
vl lv	Altivec : c、d、i、o、u、x、X のいずれかが後ろに続く場合、vector int、vector unsigned int、vector bool int。
	フラグ
-	変換は左に揃える。
+	数値の場合、変換は符号 (+ または -) が前に付加される。デフォルトでは、負の数の値のみマイナス符号 (-) が前に付加される。
space	変換の最初の文字が符号文字でない場合、1 つの空白が前に付加される。空白のフラグはプラスフラグ (+) と組み合わせる場合、プラスの符号フラグは常に符号を数値の前に付加するため効果がない。

#	c, d, i, u の変換の場合、# フラグは効果がない。s の変換の場合、Pascal 文字列のポインタは文字列として出力される。o 変換の場合、# フラグでは、ひとつの 0 が前に付加される。このフラグで x 変換の場合、0x を前に付加する。e, E, f, g, G 変換の場合、# フラグは、出力に 10 進小数点を付け加える。g と G の変換では、削除されない 10 進小数点の後にゼロが並ぶ。
0	このフラグは、変換の左側にゼロを埋め込む。これは、d, i, o, u, x, X, e, E, f, g, G 変換に適用される。通常は空白文字である部分に、符号の後の先頭の 0、基数を示す文字が並ぶ。マイナスの符号フラグは、0 フラグの文字に取って代わる。0 は、d, i, o, u, x, X 変換の型に対して精度幅とともに利用される場合、無視される。
@	Altivec: このフラグは引数によって指定された文字列へのポインタを示す。この文字列はベクトル要素のセパレータをして使用される。
	変換
d	対応する引数は、符号付き 10 進数に変換される。
i	対応する引数は、符号付き 10 進数に変換される。
o	引数は、符号なし 8 進数に変換される。
u	引数は符号なし 10 進数に変換される。
x, X	引数は符号なしの 16 進数に変換される。x 変換の場合は小文字 (abcdef) が使われ、X 変換では大文字 (ABCDEF) が使われる。
n	この変換の型は、これまでの printf() による出力された項目の数を記憶する。これに対応する引数は整数型へのポインタでなければならない。
f	対応する浮動小数点の引数 (float または double) は、10 進表記で出力される。デフォルトの精度は、6 (10 進小数点以下 6 桁)。精度幅が明確に 0 の場合、10 進小数点は出力されない。
e, E	浮動小数点の引数 (float または double) は、指数表現で出力される: [-]b.aaae ± Eee。10 進小数点の前に 1 桁 (b) の数字がある。オプションの精度幅が指定されない場合、デフォルトは 10 進小数点から 6 桁である。精度幅が 0 なら 10 進小数点は出力されない。指数 (ee) は、少なくとも 2 桁である。 e 変換では、指数の前に小文字の e が付加されるのは E の変換は、大文字 E が指数の前に付加される。
g, G	g 変換では、f あるいは e 変換が使われ、G 変換では F や E の変換が使われる。変換の e (または E) は、変換された指数が -4 より小さいかあるいは精度幅より大きいときに限り利用される。精度幅では有効数字を指定する。10 進小数点以下に続く数字がなければ、小数点は出力されない。
c	対応する引数は 1 文字として出力される。
s	対応する引数、文字配列を指すポインタは文字列として出力される。文字列の出力は、ヌル文字に到達した時点で終了する。ヌル文字は出力されない。

p	対応する引数はポインタとして扱われる。引数は x 変換の書式を使って出力される。
	CodeWarrior での拡張
#s	対応する引数、すなわち Pascal 文字列へのポインタは、文字列として出力される。Pascal 文字列はバイト長で指定された数字の文字が連続するバイト長である。 注意：この変換は ANSI C ライブラリに対しての拡張で、他の書式のバリエーションには同じ方法で適応されない。

戻り値 `printf()` は `fprintf()`、`sprintf()`、`vfprintf()`、`vprintf()` と同様、出力が成功した場合は引数の数を返し、失敗した場合は負の値を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「fprintf」\(p187\)](#)、[「sprintf」\(p231\)](#)、[「vfprintf」\(p237\)](#)、[「vprintf」\(p239\)](#)、[「vsprintf」\(p240\)](#)

例 25.25 `printf()` の使用例

```
#include <stdio.h>

int main(void)
{
    int i = 25;
    char c = 'M';
    short int d = 'm';
    static char s[] = "Metrowerks!";
    static char pas[] = "\pMetrowerks again!";
    float f = 49.95;
    double x = 1038.11005;
    int count;
    printf("%s printf() demonstration:\n%n", s, &count);
    printf("The last line contained %d characters\n",count);
    printf("Pascal string output: %#20s\n", pas);
    printf("%-4d %x %06x %-5o\n", i, i, i, i);
    printf("%*d\n", 5, i);
    printf("%4c %4u %4.10d\n", c, c, c);
    printf("%4c %4hu %3.10hd\n", d, d, d);
    printf("$%5.2f\n", f);
    printf("%5.2f\n%6.3f\n%7.4f\n", x, x, x);
    printf("%*.*f\n", 8, 5, x);

    return 0;
}
```

Output:
Metrowerks! printf() demonstration:
The last line contained 36 characters
Pascal string output: Metrowerks again!

```

25    19 000019 31
      25
      M    77 0000000077
      m    109 0000000109
$49.95
1038.11
1038.110
1038.1101
1038.11005

```

例 25.26 Altivec Printf 拡張の使用例

```

#include <stdio.h>

int main(void)
{
    vector signed char s =
        (vector signed char)(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16);
    vector unsigned short us16 =
        (vector unsigned short)('a','b','c','d','e','f','g','h');
    vector signed int sv32 =
        (vector signed int)(100, 2000, 30000, 4);
    vector signed int vs32 =
        (vector signed int)(0, -1, 2, 3);
    vector float f1t32 =
        (vector float)(1.1, 2.22, 3.3, 4.444);

    printf("s = %vd\n", s);

    printf("s = %,vd\n", s);

    printf("vector=%@vd\n", "\nvector=", s);

    // c specifier so no space is added.
    printf("us16 = %vhc\n", us16);

    printf("sv32 = %,5lvd\n", sv32);

    printf("vs32 = 0x%@.8lvX\n", "", 0x", vs32);

    printf("flt32 = %,5.2vf\n", f1t32);

    return 0;
}

```

The Result is:

```

s = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
s = 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

```



```
vector=1
vector=2
vector=3
vector=4
vector=5
vector=6
vector=7
vector=8
vector=9
vector=10
vector=11
vector=12
vector=13
vector=14
vector=15
vector=16
us16 = abcdefgh
sv32 = 100, 2000, 30000, 4
vs32 = 0x00000000, 0xFFFFFFFF, 0x00000002, 0x00000003
flt32 = 1.10, 2.22, 3.30, 4.44
```

putc

ストリームに 1 文字書きます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int putc(int c, FILE *stream);`

引数 この関数の引数です。

c	int	ファイルへ書き込む文字
stream	FILE *	FILE ストリームへのポインタ

注意点 `putc()` 関数は `c` をストリームに出力し、ストリームのファイル位置指標を前進させます。

`putc()` は、マクロで書かれているということを除けば、`fputc()` と同じように動作します。

ファイルを更新モード (+) でオープンしている場合に書き込み操作と読み取り操作がストリームバッファをフラッシュする操作で区切られていなければ、ファイルの読み取りはできず、またファイルへの書き込みもできません。これは `fflush()` 関数、またはファイル位置操作 (`fseek()`、`fsetpos()`、`rewind()`) のいずれかで可能です。

戻り値 `putc()` は成功した場合、書き込んだ文字を返し、失敗した場合、EOF を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「putc」\(p191\)](#)、[「putchar」\(p218\)](#)

例 25.27 `putc()` 使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    static char filename[] = "checkputc";
    static char test[] = "flying fish and quail eggs";
    int i;

    // 出力用の新しいファイルを生成
    if (( f = fopen(filename, "w")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // putc() を使って一度に 1 文字
    // 文字配列 test へ出力
    for (i = 0; test[i] > 0; i++)
        putc(test[i], f);

    // ファイルをクローズ
    fclose(f);

    return 0;
}
```

putc

`stdout` に 1 文字書き込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
int putchar(int c);
```

引数 この関数の引数です。

c int stdout へ書き込む文字

注意点 putchar() は stdout に文字 c を書き込みます。

戻り値 putchar() は成功すれば c を返し、失敗すれば EOF を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「fputc」\(p191\)](#)、[「putc」\(p217\)](#)

例 25.28 putchar() の使用例

```
#include <stdio.h>

int main(void)
{
    static char test[] = "running jumping walking tree\n";
    int i;

    // ヌル文字が見つかるまで
    // 一度に 1 文字テスト文字を出力する
    for (i = 0; test[i] != '\0'; i++)
        putchar(test[i]);

    return 0;
}
```

Output:
running jumping walking tree

puts

stdout に 1 文字列書き込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <stdio.h>
int puts(const char *s);

引数 この関数の引数です。

s const char * stdout へ書き込む文字列

注意点 puts() 関数は、stdout に 1 文字列をヌル文字のところまで書き込みます。ヌル文字は含みません。この関数は改行 ('\n') を出力に付け加えます。

戻り値 puts() は成功した場合はゼロを返し、失敗した場合は非ゼロを返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「fputs」\(p193\)](#)

例 25.29 puts() の使用例

```
#include <stdio.h>

int main(void)
{
    static char s[] = "car bus metro werks";
    int i;

    // 文字列を 10 回出力する
    for (i = 0; i < 10; i++)
        puts(s);

    return 0;
}
```

```
Output:
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
```

remove

ファイルを削除します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
int remove(const char *filename);
```

引数 この関数の引数です。

filename	const char *	削除するファイルの名前
----------	--------------	-------------

注意点 `remove()` は、引数 `filename` で指定された名前のファイルを削除します。

戻り値 `remove()` はファイルの削除に成功した場合は0を返し、失敗した場合は非ゼロの値を返します。

参照 [「fopen」\(p184\)](#)、[「rename」\(p221\)](#)

例 25.30 `remove()` の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char filename[40];

    // ユーザからファイル名を得る
    printf("Enter the name of the file to delete.\n");
    gets(filename);

    // ファイルを削除する
    if (remove(filename) != 0) {
        printf("Can't remove %s.\n", filename);
        exit(1);
    }

    return 0;
}
```

rename

ファイルの名前を変更します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
int rename(const char *old, const char *new);
```

引数 この関数の引数です。

old	const char *	古いファイル名
new	const char *	新しいファイル名

注意点 `rename()` 関数は、引数 `old` によって指定されたファイルの名前を引数 `new` で指定した名前に変更します。

戻り値 `rename()` は失敗した場合は非ゼロを返し、成功した場合はゼロを返します。

参照 [「freopen」\(p196\)](#)、[「remove」\(p220\)](#)

例 25.31 `rename()` の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char oldname[50]; // 現在のファイル名
    char newname[50]; // 新しいファイル名

    // ユーザから現在のファイル名を得る
    printf("Please enter the current filename.\n");
    gets(oldname);

    // ユーザから現在のファイル名を得る
    printf("Please enter the new filename.\n");
    gets(newname);

    // 古い名前から新しい名前に付け替える
    if (rename(oldname, newname) != 0) {
        printf("Can't rename %s to %s.\n", oldname,
            newname);
        exit(1);
    }

    return 0;
}
```

Output:
Please enter the current filename.
metrowerks
Please enter the new filename.
itwerks

rewind

ファイル位置指標をファイルの始まりにリセットします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`void rewind(FILE *stream);`

引数 この関数の引数です。

stream FILE * FILE ストリームへのポインタ

注意点 `rewind()` 関数はストリームのファイル指標の位置を次の読み書き操作がファイルの始まりであるように設定します。それまでの `ungetc()` の呼び出しを元に戻し、ストリームのファイルの終わりやエラー状態をクリアします。

注意： embedded/RTOS システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

参照 [「fseek」\(p201 \)](#)、[「fsetpos」\(p204 \)](#)

例 25.32 `rewind()` の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    char filename[80], buf[80];

    // ユーザからファイル名を得る
    printf("Enter a filename to read.\n");
    gets(filename);

    // 入力用のファイルをオープン
    if (( f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    printf("Reading first line twice.\n");

    // ファイル位置指標を
    // ファイルの始まりに移動
    rewind(f);
    // 最初の行を読み込む
```

```
fgets(buf, 80, f);
printf("Once: %s\n", buf);

// ファイル位置指標を
// ファイルの始まりに移動
rewind(f);

// 再び最初の行を読み込む
fgets(buf, 80, f);
printf("Twice: %s\n", buf);

// ファイルをクローズ
fclose(f);

return 0;
}
```

Output:
Enter a filename to read.
itwerks
Reading first line twice.
Once: flying fish and quail eggs
Twice: flying fish and quail eggs

scanf

書式化してテキストを読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int scanf(const char *format, ...);`

引数 この関数の引数です。

format const char * 書式化した文字列

注意点 この関数は以下のターゲットに対して互換性があります。

Scanf 制御文字列と変換指定子

書式引数は、通常のテキストと何も書かれていない空白（空白、タブ、改行）と変換仕様です。通常のテキストは入力ストリームでマッチされなければならない文字を指定します。何も書かれていない空白の文字は、何か書かれている文字が見つかるまで空白文字を読み

飛ばすことを示しています。変換仕様は、入力ストリーム中のどの文字が変換され記憶されるのかを指定しています。

変換仕様は、書式中に現れる順番のマッチング引数を持つ必要があります。マッチングの変換仕様の引数は、関連のある型へのポインタでなければなりません。これはメモリにデータを記憶するためです。

変換仕様は、前に付加するパーセントの符号 (%)、オプションの最大幅あるいは割付の抑制、そして変換の型で終わります。1 個のパーセントの符号は、書式中では 2 つにすることによって読み飛ばすことができます (入力ストリーム中では、%% が単体の % を表します)。

オプションの幅は、入力範囲の最大幅を指定する 10 進数です。scanf () は一度の変換で、指定された幅以上の文字は読み込みません。

オプションの割付の抑制文字 (*) は、1 個の項目を読み込むが割り付けられないよう、項目を読み飛ばすために利用することができます。割付抑制を伴う変換仕様は対応する引数を持つてはいけません。

最後の文字、それは変換の型で要求された変換の種類を指定するものです。[表 25.5 \(p225 \)](#) に、変換の型の文字について説明します。

Scanf に対する MSL Altivec 拡張

MSL は、標準 printf 関数への Altivec 拡張をサポートしています。

% の後と、指定子の前のセパレータ引数には、どんな文字でも使用できます。@ シンボルも使用できます。@ シンボルは非 Motorola 拡張子であり、指定された文字列を指定子として使用します。

'c' 指定子は、任意の char をセパレータとして使うことができますが、'-','+', '#',' ' は使用できません。

[「Altivec Scanf 拡張の使用例」\(p227 \)](#) を参照してください。

表 25.5 書式化した入力関数の書式修飾子の型

修飾子	説明
	修飾子のサイズ
h	フラグは対応する変換修飾子が short int 型か unsigned short int 型であることを示す。
l	整数の変換修飾子を用いた場合、l フラグは long int 型か unsigned long int 型を指す。浮動小数点変換修飾子を用いた場合、l フラグは double 型であることを示す。
ll	整数変換修飾子と一緒に使われた場合、ll フラグは long long または unsigned long long 型を示す。
L	L フラグは対応する float 変換修飾子が long double 型であることを示す。

v	Altivec: c、d、i、o、u、x、X のいずれかが後ろに続く場合、vector bool char、vector signed char、または vector unsigned char。 f が後ろに続く場合、vector float。
vh hv	Altivec: c、d、i、o、u、x、X のいずれかが後ろに続く場合、vector short、vector unsigned short、vector bool short、vector pixel。
vl lv	Altivec: c、d、i、o、u、x、X のいずれかが後ろに続く場合、vector long、vector unsigned long、vector bool。
	変換修飾子
d	1 個の 10 進整数を読み込む。
i	1 個の 10 進数、8 進数あるいは 16 進数の整数を読み込む。整数はプラスやマイナスの符号 (+, -) 8 進数の数に関しては 0、16 進数の数に関しては 0x や 0X を前に付け加えることが可能。
o	1 個の 8 進数の整数を読み込む。
u	1 個の符号なしの 10 進数の整数を読み込む。
x, X	1 個の 16 進数の整数を読み込む。
e, E, f, g, G	1 個の浮動小数点数を読み込む。数は通常の 10 進数の書式 (例えば、3456.483) あるいは指数表現が可能。 ([-]b.aaae 7 dd)。
s	1 個の文字列を読み込む。入力文字列は空白に到達した場合、または最大幅に到達した場合、終了と見なす。ヌル文字を配列の終わりに付け加える。
c	1 個の文字を読み込む。空白文字は読み飛ばされず、この変換の型を用いて読み込む。
p	1 個のポインタのアドレスを読み込む。入力書式は printf() での p 変換による出力と同じ。
n	この変換型は、入力ストリームから読み込まず、対応する引数中の読み込んだ文字数を記憶する。
[scanset]	入力ストリームキャラクタを読み取り、scanset で指定されたフィルタを事項する。 「Scanset」(p199) を参照。

戻り値 scanf() は、読み込みに成功した項目の数を返し、変換の型が引数にマッチしない場合やファイルの終わりに到達した場合は EOF を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「fscanf」\(p197\)](#)、[「sscanf」\(p232\)](#)

例 25.33 scanf() の使用例

```
#include <stdio.h>

int main(void)
{
    int i;
    unsigned int j;
    char c;
    char s[40];
    double x;

    printf("Enter an integer surrounded by ! marks\n");
    scanf("!!d!", &i);
    printf("Enter three integers\n");
    printf("in hexadecimal, octal, or decimal.\n");
    // 3つの整数が読み込まれるが、
    // 終わりの2つがiとjに割り付けられたことに注意
    scanf("%*i %i %ui", &i, &j);

    printf("Enter a character and a character string.\n");
    scanf("%c %10s", &c, s);

    printf("Enter a floating point value.\n");
    scanf("%lf", &x);

    return 0;
}
```

Output:

```
Enter an integer surrounded by ! marks
!!94!
Enter three integers
in hexadecimal, octal, or decimal.
1A 6 24
Enter a character and a character string.
Enter a floating point value.
A
Sounds like 'works'!
3.4
```

例 25.34 Altivec Scanf 拡張の使用例

```
#include <stdio.h>

int main(void)
{
    vector signed char v8, vs8;
```

```
vector unsigned short v16;
vector signed long v32;
vector float vf32;

sscanf("1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16", "%vd", &v8);
sscanf("1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", "%,vd", &vs8);
sscanf("abcdefgh", "%vhc", &v16);
sscanf("1, 4, 300, 400", "%,3lvd", &v32);
sscanf("1.10, 2.22, 3.333, 4.4444", "%,5vf", &vf32);

return 0;
}
```

The Result is:

```
v8 = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16;
vs8 = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16;
v16 = 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'
v32 = 1, 4, 300, 400
vf32 = 1.1000, 2.2200, 3.3330, 4.4444
```

setbuf

ストリームのバッファサイズを変更します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`void setbuf(FILE *stream, char *buf);`

引数 この関数の引数です。

stream	FILE *	FILE ストリームへのポインタ
buf	char *	入力、出力用のバッファ

注意点 `setbuf()` は、プログラマにストリームのバッファサイズを設定することを許可します。それはストリームがオープンされた後で、データが読み書きされる前に行われなければならない。

関数は、引数 `buf` によって指し示される配列を作成します。`buf` は、ヌルポインタかあるいは `stdio.h` で定義された `BUFSIZ` サイズの配列を指します。

ヌルポインタの場合、ストリームをバッファ化しません。

参照 [「setvbuf」\(p229\)](#)、[「malloc」\(p260\)](#)

例 25.35 setbuf() の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    char name[80];

    // ユーザからファイル名を得る
    printf("Enter the name of the file to write to.\n");
    gets(name);

    // 出力用の新しいファイルを生成
    if ( (f = fopen(name, "w")) == NULL) {
        printf("Can't open file %s.\n", name);
        exit(1);
    }

    setbuf(f, NULL); // バッファリングの中止

    // このテキストはバッファリングすることなしに、
    // 直接ファイルに送られる
    fprintf(f, "Buffering is now off\n");
    fprintf(f, "for this file.\n");

    // ファイルをクローズ
    fclose(f);

    return 0;
}
```

Output:
Enter the name of the file to write to.
bufftest

setvbuf

ストリームのバッファリング計画を変更します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <stdio.h>
int setvbuf(FILE *stream, char *buf, int mode,

```
size_t size);
```

引数 この関数の引数です。

stream	FILE *	FILE ストリームへのポインタ
buf	char *	入力、出力用のバッファ
mode	int	バッファモデル
size	size_t	バッファのサイズ

注意点 `setvbuf()` は、ストリームで使用されるバッファリング計画とサイズの操作を許可します。この関数は、ストリームがオープンしてから読み書きがなされる前までに、呼び出さなければなりません。

引数 `buf` は文字配列に対してのポインタです。引数 `size` は `buf` によって指し示された文字配列のサイズを指定します。もっとも効果的なバッファサイズは `stdio.h` で定義されている `BUFSIZ` の 2 倍です。

`buf` がヌルポインタである場合、オペレーティングシステムはそれ自身のバイトサイズのバッファを生成します。

`mode` 引数ではストリームに使用されるバッファリング計画を指定します。`mode` には、`stdio.h` の 3 つの値、`_IOFBF`、`_IOLBF`、`_IONBF` のいずれか 1 つを指定します。

- `_IOFBF` : ストリームをバッファ化することを指定する
- `_IOLBF` : ストリームをラインバッファ化することを指定する
- `_IONBF` : ストリームをバッファ化しないことを指定する

戻り値 `setvbuf()` は成功した場合はゼロを返し、失敗した場合は非ゼロを返します。

参照 [「setbuf」\(p228\)](#)、[「malloc」\(p260\)](#)

例 25.36 setvbuf() の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    char name[80];

    // ユーザからファイル名を得る
    printf("Enter the name of the file to write to.\n");
    gets(name);

    // 出力用の新しいファイルを生成
    if ( (f = fopen(name, "w")) == NULL) {
        printf("Can't open file %s.\n", name);
```

```
        exit(1);
    }

    setvbuf(f, NULL, _IOLBF, 0); // line buffering
    fprintf(f, "This file is now\n");
    fprintf(f, "line buffered.\n");

    // ファイルをクローズ
    fclose(f);

    return 0;
}
```

Output:
Enter the name of the file to write to.
buffy

sprintf

文字列用配列を書式化します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int sprintf(char *s, const char *format, ...);`

引数 この関数の引数です。

s	char *	書き込む文字列
format	const char *	書式化した文字列

注意点 `sprintf()` 関数は基本的には `printf()` と同じように動作し、引数が追加されています。出力は `stdout` に送られる代わりに、`s` によって指定された文字配列に記憶されます。この関数の出力された文字列は、ヌル文字で終わります。

出力制御文字列と変換指定子に関する仕様は「[出力制御文字列と変換指定子](#)」(p187)を参照してください。

戻り値 `sprintf()` は、`s` に割り当てられた文字数（ヌル文字を含まない）を返します。

参照 「[拡張文字とバイト文字のストリーム方向](#)」(p171)、 「[fprintf](#)」(p187)、 「[printf](#)」(p211)

例 25.37 sprintf() の使用例

```
#include <stdio.h>

int main(void)
{
    int i = 1;
    static char s[] = "Metrowerks";
    char dest[50];

    sprintf(dest, "%s is number %d!", s, i);
    puts(dest);

    return 0;
}
```

Output:
Metrowerks is number 1!

sscanf

文字列中の書式化されたテキストを読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
int sscanf(char *s, const char *format, ...);
```

引数 この関数の引数です。

s	char *	スキャンする文字列
format	const char *	書式化した文字列

注意点 sscanf() は、scanf() と同じように動作しますが、stdin の代わりに引数 s で指された文字配列からその入力を読み込みます。s で指定される文字配列はヌルで終わる必要があります。

入力制御文字列と変換指定子の仕様は「[入力制御文字列と変換指定子](#)」(p197) を参照してください。scansets の使い方については「[Scanset](#)」(p199) を参照してください。

戻り値 scanf() は、読み込みと変換が成功すれば項目の数を返し、文字列の終わりに到達した場合、または変換仕様がその引数にマッチしない場合は EOF を返します。

参照 「[拡張文字とバイト文字のストリーム方向](#)」(p171)、「[fscanf](#)」(p197)、「[scanf](#)」(p224)

例 25.38 sscanf() の使用例

```
#include <stdio.h>

int main(void)
{
    static char in[] = "figs cat pear 394 road 16!";
    char s1[20], s2[20], s3[20];
    int i;

    //
    // 単語 figs, cat, road と整数値 16 を、in から読み込み
    // s1, s2, s3, i にそれぞれ格納する
    //
    sscanf(in, "%s %s pear 394 %s %d!", s1, s2, s3, &i);
    printf("%s %s %s %d\n", s1, s2, s3, i);

    return 0;
}
```

Output:
figs cat road 16

tmpfile

テンポラリファイルをオープンします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
FILE *tmpfile(void);
```

注意点 tmpfile() 関数は、1 個のバイナリファイルを生成してオープンします。このファイルはクローズされた場合やプログラムが終了した場合、自動的に削除されます。

戻り値 tmpfile() は成功した場合、テンポラリファイルの FILE 変数へのポインタを返します。失敗した場合、ヌルポインタ (NULL) を返します。

参照 [「fopen」\(p184\)](#)、[「tmpnam」\(p234\)](#)

例 25.39 tmpfile() の使用例

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main(void)
{
    FILE *f;

    // 出力用の新しいテンポラリファイルを生成
    if ( (f = tmpfile()) == NULL) {
        printf("Can't open temporary file.\n");
        exit(1);
    }

    // テンポラリファイルにテキストを出力
    fprintf(f, "watch clock timer glue\n");

    // fclose() を使って、テンポラリファイルの
    // クローズと削除
    fclose(f);

    return 0;
}
```

tmpnam

唯一のテンポラリファイル名を生成します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
char *tmpnam(char *s);
```

引数 この関数の引数です。

s char * テンポラリファイル名

注意点 tmpnam() 関数は、現存するファイル名と一致しない有効なファイル名の文字列を生成します。プログラムは、tmpnam() が生成した唯一のファイル名を使い果たす前に、TMP_MAX 回まで関数を呼ぶことができます。TMP_MAX マクロは stdio.h で定義されています。

引数 s はヌルポインタかあるいは文字配列へのポインタです。文字配列は少なくとも L_tmpnam の文字の長さが必要です。新しいテンポラリファイル名をこの配列に記憶します。L_tmpnam マクロは stdio.h で定義されます。

s が NULL であった場合、tmpnam() は関数を呼んだプログラムによって変更可能な内部の static 変数のポインタを返します。

`tmpfile()` とは異なり、`tmpnam()` 関数によって生成されたファイル名のファイルはクローズ時に自動的に削除できません。

戻り値 `tmpnam()` は、他と一致しない唯一のファイル名が入った文字配列へのポインタを返します。`s` がヌルポインタ (`NULL`) であった場合、ポインタは内部の `static` 変数を指します。`s` が文字配列を指していれば、`tmpnam()` は同じポインタを返します。

参照 [「fopen」\(p184\)](#)、[「tmpfile」\(p233\)](#)

例 25.40 `tmpnam()` の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    char *tempname;
    int c;

    // 唯一のファイル名を得る
    tempname = tmpnam("tempwerks");

    // 出力用の新しいファイルを生成
    if ( (f = fopen(tempname, "w")) == NULL) {
        printf("Can't open temporary file %s.\n", tempname);
        exit(1);
    }

    // ファイルにテキストを出力
    fprintf(f, "shoe shirt tie trousers\n");
    fprintf(f, "province\n");

    // ファイルをクローズ
    fclose(f);

    // ファイルの削除
    remove(tempname);

    return 0;
}
```

`ungetc`

ストリーム中へ 1 文字戻します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int ungetc(int c, FILE *stream);`

引数 この関数の引数です。

c	int	ファイルへ返す文字列
stream	FILE *	FILE ストリームへのポインタ

注意点 `ungetc()` 関数は、ストリームのバッファに文字 `c` を戻す。次の読み込み操作は、`ungetc()` で戻された文字を読む。読み込み操作が実行されるまで、バッファへは 1 個の文字を戻すことが可能です。

`fseek()`、`fsetpos()`、`rewind()` 操作が実行された時点で関数の効果がなくなります。

戻り値 `ungetc()` は成功した場合 `c` を返し、失敗した場合 EOF を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「fseek」\(p201\)](#)、[「fsetpos」\(p204\)](#)、[「rewind」\(p222\)](#)

例 25.41 `ungetc()` の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    int c;

    // 入出力用の新しいファイルを生成
    if ( (f = fopen("myfoo", "w+")) == NULL) {
        printf("Can't open myfoo.\n");
        exit(1);
    }

    // ファイルへテキストを出力
    fprintf(f, "The quick brown fox\n");
    fprintf(f, "jumped over the moon.\n");

    // ファイル位置指標を、
    // ファイルの始めに動かす
    rewind(f);

    printf("Reading each character twice.\n");
```

```
// 1文字読み込む
while ( (c = fgetc(f)) != EOF) {
    putchar(c);
// ストリームへ文字を戻す
    ungetc(c, f);
    c = fgetc(f); // 再び同じ文字を読み込む
    putchar(c);
}

fclose(f);

return 0;
}
```

fprintf

ストリームに書式化された出力を書き込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int fprintf(FILE *stream,`
`const char *format, va_list arg);`

引数 この関数の引数です。

stream	FILE *	FILE ストリームへのポインタ
format	const char *	書式化した文字列

注意点 `fprintf()` 関数は `fprintf()` 関数と同じように動作します。`fprintf()` では可変な引数リストが採用されていない代わりに、`fprintf()` は `stdarg.h` ヘッダファイルの `va_start()` マクロによって処理された `va_list` 型の配列を受け取ります。

注意： `embedded/RTOS` システム上ではこの関数は `stdin`、`stdout`、`stderr` ファイル用にのみ実装されています。

出力制御文字列、変換指定子に関する仕様は「[出力制御文字列と変換指定子](#)」(p187) を参照してください。

戻り値 `fprintf()` は書かれた文字の数を返し、失敗した場合は `EOF` を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「fprintf」\(p187\)](#)、[「printf」\(p211\)](#)、[「stdarg.h の概要」\(p163\)](#)

例 25.42 vfprintf() の使用例

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>

int fpr(FILE *, char *, ...);

int main(void)
{
    FILE *f;
    static char name[] = "foo";
    int a = 56, result;
    double x = 483.582;

    // 出力用の新しいファイルの生成
    if ((f = fopen(name, "w")) == NULL) {
        printf("Can't open %s.\n", name);
        exit(1);
    }

    // ファイルへ可変な引数の数を書式化して
    // 出力します。
    result = fpr(f, "%10s %4.4f %-10d\n", name, x, a);

    // ファイルをクローズする

    fclose(f);

    return 0;
}

// fpr() は、vfprintf() 関数を使って
// ストリームへの可変な引数の数を
// 書式化して出力する
int fpr(FILE *stream, char *format, ...)
{
    va_list args;
    int retval;

    va_start(args, format); // 書式化して出力する
    retval = vfprintf(stream, format, args);
    // 引数の出力
    va_end(args); // スタックのクリア
```

```
    return retval;
}
```

vprintf

stdout に書式化された出力を書き込みます

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdio.h>`
`int vprintf(const char *format, va_list arg);`

引数 この関数の引数です。

format	const char *	書式化した文字列
arg	va_list	変数の引数リスト

注意点 `vprintf()` 関数は `printf()` 関数と同じように動作します。`fprintf()` では可変な引数リストが採用されていない代わりに、`vprintf()` は `stdarg.h` ヘッダファイルの `va_start()` マクロによって処理された `va_list` 型の配列を受け取ります。

出力制御文字列と変換指定子に関する仕様は「[出力制御文字列と変換指定子](#)」(p187) を参照してください。

戻り値 `vprintf()` は書かれた文字数を返し、失敗した場合は負の値を返します。

参照 「[拡張文字とバイト文字のストリーム方向](#)」(p171)、 「[fprintf](#)」(p187)、 「[printf](#)」(p211)、 「[stdarg.h の概要](#)」(p163)

例 25.43 vprintf() の使用例

```
#include <stdio.h>
#include <stdarg.h>

int pr(char *, ...);

int main(void)
{
    int a = 56;
    double f = 483.582;
    static char s[] = "Metrowerks";

    // stdout に対して可変な引数の数を出力
    pr("%15s %4.4f %-10d*\n", s, f, a);
}
```

```
        return 0;

    }

    // pr() は vprintf() を使って stdout に
    // 可変な引数の数を書式化して出力する
    int pr(char *format, ...)
    {
        va_list args;
        int retval;
        va_start(args, format); // 引数の準備
        retval = vprintf(format, args);
        va_end(args); // スタックのクリア
        return retval;
    }
```

```
Output:
Metrowerks 483.5820 56          *
```

vsprintf

文字列に書式化された出力を書き込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdio.h>
int vsprintf(char *s,
             const char *format, va_list arg);
```

引数 この関数の引数です。

s	char *	書き込む文字列
format	const char *	書式化した文字列
arg	va_list	変数の引数リスト

注意点 vsprintf() 関数は、printf() 関数と同じように動作します。fprintf() では可変な引数リストが採用されていない代わりに、vsprintf() は stdarg.h ヘッダファイルの va_start() マクロによって処理された va_list 型の配列を受け取ります。

出力制御文字列と変換指定子に関する仕様は「[出力制御文字列と変換指定子](#)」(p187) を参照してください。

戻り値 vsprintf() は s に書き込まれた文字の数を返します。失敗した場合 EOF を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p171\)](#)、[「printf」\(p211\)](#)、[「sprintf」\(p231\)](#)、[「stdarg.h の概要」\(p163\)](#)

例 25.44 vsprintf() の使用例

```
#include <stdio.h>
#include <stdarg.h>

int spr(char *, char *, ...);

int main(void)
{
    int a = 56;
    double x = 1.003;
    static char name[] = "Metrowerks";
    char s[50];

    // 文字配列 s に可変の引数の数を
    // 書式化して転送する
    spr(s, "%10s\n %f\n %-10d\n", name, x, a);
    puts(s);

    return 0;
}

// spr() は sprintf() 関数を用いて、
// 文字配列に可変な引数の数を
// 書式化して転送する
int spr(char *s, char *format, ...)
{
    va_list args;
    int retval;

    va_start(args, format); // 引数の準備
    retval = vsprintf(s, format, args);
    va_end(args); // スタックのクリア
    return retval;
}
```

Output:
Metrowerks
1.003000
56

第 26 章 stdlib.h

Tstdlib.h ヘッダファイルは、文字列の変換、疑似乱数生成、メモリ管理、環境との通信、検索と並び替え、マルチバイトの文字変換、整数値の計算に密接に関連した関数のグループを提供します。

stdlib.h の概要

stdlib.h ヘッダファイルには以下の関数があります。

以下は文字変換の関数です。

[「atof」\(p248\)](#)

[「strtod」\(p266\)](#)

以下は疑似乱数関数です。

[「rand」\(p264\)](#)

[「srand」\(p266\)](#)

以下はメモリ管理の関数です。

[「calloc」\(p253\)](#)

[「free」\(p257\)](#)

[「malloc」\(p260\)](#)

[「realloc」\(p265\)](#)

[「vec_calloc」\(p270\)](#)

[「vec_free」\(p271\)](#)

[「vec_malloc」\(p271\)](#)

[「vec_realloc」\(p271\)](#)

以下は環境との通信の関数です。

[「abort」\(p244\)](#)

[「atexit」\(p246\)](#)

[「exit」\(p256\)](#)

[「getenv」\(p258\)](#)

[「system」\(p270\)](#)

以下は検索、並び替えの関数です。

[「bsearch」\(p250 \)](#)

[「qsort」\(p263 \)](#)

以下はマルチバイト変換の関数です。地域設定情報で指定されたマルチバイトの文字を wchar_t 型の文字 (stddef.h で定義) に変換します。

[「mblen」\(p261 \)](#)

[「mbstowcs」\(p261 \)](#)

[「mbtowc」\(p262 \)](#)

[「wcstombs」\(p272 \)](#)

[「wctomb」\(p273 \)](#)

以下は整数計算の関数です。

[「abs」\(p245 \)](#)

[「div」\(p255 \)](#)

[「labs」\(p259 \)](#)

[「ldiv」\(p260 \)](#)

stdlib.h 関数の多くは stdlib.h で定義される size_t 型と NULL マクロを使います。

abort

プログラムの異常終了です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdlib.h>
void abort(void)
```

引数 なし。

注意点 abort() 関数は、オペレーティングシステムに復帰するため、SIGABRT シグナルを発生させてプログラムを終了します。

abort() 関数は、プログラマがインストールしたシグナルハンドラが通常の復帰の代わりに longjmp() を使った場合、プログラムを終了しません。

参照 [「assert」\(p23 \)](#) [「longjmp」\(p129 \)](#) [「raise」\(p137 \)](#) [「signal」\(p135 \)](#) [「atexit」\(p246 \)](#) [「exit」\(p256 \)](#)

例 26.1 abort() の使用例

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    char c;

    printf("Aborting the program.\n");
    printf("Press return.\n");

    // リターンキーが押されるまで待つ
    c = getchar();

    // プログラムの異常終了
    abort();

    return 0;
}
```

Output:
Aborting the program.
Press return.

abs

整数の絶対値を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <stdlib.h>
int abs(int i);

引数 この関数の引数です。

i	int	計算されている値
---	-----	----------

戻り値 abs() は引数の絶対値を返します。最も小さい負の数の 2 の補数表現は、整数の絶対値表現とはマッチしないことに注意してください。

参照 [「fabs」\(p92\)](#)、[「labs」\(p259\)](#)

例 26.2 abs() の使用例

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    int i = -20;
    long int j = -48323;

    printf("Absolute value of %d is %d.\n", i, abs(i));
    printf("Absolute value of %ld is %ld.\n", j, labs(j));

    return 0;
}
```

Output:
Absolute value of -20 is 20.
Absolute value of -48323 is 48323.

atexit

プログラムが終了する場合に実行する関数をインストールします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`
`int atexit(void (*func) void);`

引数 この関数の引数です。

func void * 終了時に実行する関数

注意点 atexit() 関数は、func によって指された関数をリストに追加します。exit() が呼ばれると、リスト上の各関数は atexit() によって登録された順番とは逆の順番で呼ばれます。リスト上の全ての関数が呼ばれた後、exit() がプログラムを終了します。

例えば、stdio.h ライブラリはそれ自身の exit 関数を atexit() によって登録します。この関数はすべてのバッファをフラッシュし、全てのオープンされたストリームをクローズします。

戻り値 新しい exit 関数が登録されていれば、atexit() は成功した場合はゼロを返し、失敗した場合は非ゼロの値を返します。

参照 [「exit」\(p256\)](#)

例 26.3 atexit() の使用例

```
#include <stdlib.h>
#include <stdio.h>

// プロトタイプ
void first(void);
void second(void);
void third(void);

int main(void)
{
    atexit(first);
    atexit(second);
    atexit(third);

    printf("exiting program\n\n");
    return 0;
}

void first(void)
{
    int c;

    printf("First exit function.\n");
    printf("Press return.\n");
    // リターンキーが押されるまで待つ
    c = getchar();
}

void second(void)
{
    int c;

    printf("Second exit function.\n");
    printf("Press return.\n");
    c = getchar();
}

void third(void)
{
    int c;

    printf("Third exit function.\n");
    printf("Press return.\n");
    c = getchar();
}
```

Output:
Third exit function.

Press return.

Second exit function.
Press return.

First exit function.
Press return.

atof

文字配列を数値に変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`
`double atof(const char *nptr);`

引数 この関数の引数です。

`nptr` `const char *` 変換される文字列

注意点 `atof()` 関数は、`nptr` で指された文字配列を `double` 型の浮動小数点の値に変換します。

この関数は最初の空白文字をスキップします。

この関数は、変換された値がそれぞれの型で表現できない場合、大域変数 `errno` に `ERANGE` を設定します。

戻り値 `atof()` は、`double` 型の浮動小数点の値を返します。

参照 [「atoi」\(p249 \)](#)、[「atol」\(p249 \)](#)、[「errno」\(p49 \)](#)、[「scanf」\(p224 \)](#)

例 26.4 atof()、atoi()、atol() の使用例

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    int i;
    long int j;
    float f;
    static char si[] = "-493", sli[] = "63870";
    static char sf[] = "1823.4034";
```



```
        f = atof(sf);
        i = atoi(si);
        j = atol(sli);

        printf("%f %d %ld\n", f, i, j);

        return 0;
    }
```

Output:
1823.403400 -493 63870

atoi

文字配列を数値に変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `int atoi(const char *nptr);`

引数 この関数の引数です。

`nptr` `const char *` 変換される文字列

注意点 `atoi()` 関数は、`nptr` によって指された文字配列を整数型の値に変換します。

この関数は最初の空白はスキップします。

この関数は、変換された値がそれぞれの型で表現できない場合、大域変数 `errno` に `ERANGE` を設定します。

戻り値 `atoi()` は、`int` 型の整数値を返します。

参照 [「atof」\(p248\)](#)、[「atol」\(p249\)](#)、[「errno」\(p49\)](#)、[「scanf」\(p224\)](#)

atol

文字配列を数値に変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`

```
double atof(const char *nptr);
int atoi(const char *nptr);
long int atol(const char *nptr);
```

引数 この関数の引数です。

```
nptr      const char *      変換される文字列
```

注意点 `atol()` 関数は、`nptr` によって指された文字配列を `long int` 型の整数値に変換します。
この関数は最初の空白をスキップします。

この関数は、もし変換された値がそれぞれの型で表現できない場合、大域変数 `errno` に `ERANGE` を設定します。

戻り値 `atol()` は `long int` 型の整数値を返します。

参照 [「atof」\(p248\)](#)、[「atoi」\(p249\)](#)、[「errno」\(p49\)](#)、[「scanf」\(p224\)](#)

bsearch

ソート済みの配列を効率的に検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

```
プロトタイプ #include <stdlib.h>
void *bsearch(const void *key, const void *base,
              size_t num, size_t size,
              int (*compare)(const void *, const void *))
```

引数 この関数の引数です。

```
key      const void *      検索したい項目
base     const void *      検索する配列
compare  const void *      比較に使われる関数へのポインタ
```

注意点 `bsearch()` 関数は、バイナリサーチアルゴリズムを用いて、ソート済みの配列の項目を検索します。

引数 `key` は検索したい項目を指します。

引数 `base` は、検索する配列の最初のバイト数を指します。この配列は、すでに昇順にソートされている必要があります。この順番は、引数 `compare` によって指されている関数の比較条件に基づいています。

引数 `num` は、検索のための配列要素の数を指定します。

引数 `size` は、配列要素のサイズを指定します。

引数 `compare` は、プログラマが提供する関数を指します。この関数は、配列の個々の要素と `key` を比較するために利用されます。最初の引数は `key` であり、`bsearch()` に対する最初の引数として渡されます。2 つめの引数は、配列のある要素へのポインタであり、`bsearch()` に対する 2 番目の引数として渡されます。

説明のために引数を `search_key` と `array_element` と呼ぶことにします。この比較の関数は、`search_key` と `array_element` を比較します。`search_key` と `array_element` が等しい場合、関数は、ゼロを返します。`search_key` が `array_element` より小さい場合、関数は負の値を返します。`search_key` が `array_element` より大きい場合、関数は正の値を返します。

戻り値 `bsearch()` は、`key` によって指された項目にマッチする配列中の要素を指すポインタを返します。マッチしない場合 `bsearch()` は、ヌルポインタ (`NULL`) を返します。

参照 [「qsort」\(p263\)](#)

例 26.5 bsearch の使用例

```
// 簡単な電話番号管理
// このプログラムは、名前と電話番号のリストを
// 受け取り、リストをソートして、指定された
// 名前を探す

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// 電話番号台帳の最大レコード数
#define MAXDIR 40

typedef struct
{
    char lname[15]; // key 領域 --comp() 関数を参照のこと
    char fname[15];
    char phone[15];
} DIRENTRY; // 電話番号台帳のレコード

int comp(const DIRENTRY *, const DIRENTRY *);
DIRENTRY *look(char *);
DIRENTRY directory[MAXDIR]; // 台帳そのもの
int reccount; // 入力されたレコードの数

int main(void)
{
    DIRENTRY *ptr;
    int lastlen;
```

```

char lookstr[15];

printf("Telephone directory program.\n");
printf("Enter blank last name when done.\n");

reccount = 0;
ptr = directory;
do {
    printf("\nLast name: ");
    gets(ptr->lname);
    printf("First name: ");
    gets(ptr->fname);
    printf("Phone number: ");
    gets(ptr->phone);
    if ( (lastlen = strlen(ptr->lname)) > 0) {
        reccount++;
        ptr++;
    }
} while ( (lastlen > 0) && (reccount < MAXDIR) );

printf("Thank you.  Now sorting.  . .\n");

// qsort() を使った配列のソート
qsort(directory, reccount,
        sizeof(directory[0]), (void *)comp);

printf("Enter last name to search for,\n");
printf("blank to quit.\n");
printf("\nLast name: ");
gets(lookstr);

while ( (lastlen = strlen(lookstr)) > 0) {
    ptr = look(lookstr);
    if (ptr != NULL)
        printf("%s, %s: %s\n",
                ptr->lname,
                ptr->fname,
                ptr->phone);
    elseprintf("Can't find %s.\n", lookstr);
    printf("\nLast name: ");
    gets(lookstr);
}

printf("Done.\n");

return 0;
}

int comp(const DIRENTRY *rec1, const DIRENTRY *rec2)
{

```

```
        return (strcmp((char *)rec1->lname,
                        (char *)rec2->lname));
    }

    // bsearch() を使って配列を検索
    DIRENTRY *look(char k[])
    {
        return (DIRENTRY *) bsearch(k, directory, reccount,
        sizeof(directory[0]), (void *)comp);
    }
```

Output

Telephone directory program.
Enter blank last name when done.

Last name: Mation
First name: Infor
Phone number: 555-1212

Last name: Bell
First name: Alexander
Phone number: 555-1111

Last name: Johnson
First name: Betty
Phone number: 555-1010

Last name:
First name:
Phone number:
Thank you. Now sorting. . .
Enter last name to search for,
blank to quit.

Last name: Mation
Infor, Mation: 555-1212

Last name: Johnson
Johnson, Betty: 555-1010

Last name:
Done.

calloc

対象のグループにメモリ空間を割り当てます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`
`void *calloc(size_t nmemb, size_t size);`

引数 この関数の引数です。

<code>nmemb</code>	<code>size_t</code>	要素の数
<code>size</code>	<code>size_t</code>	要素のサイズ

注意点 `calloc()` 関数は、大きさ `size` の要素の `nmemb` 個分の連続したメモリ空間を割り当てます。
メモリ空間をゼロで初期化します。

戻り値 `calloc()` は割り当てられたメモリの最初のバイトを指すポインタを返します。`calloc()`
は、メモリ空間を割り当てることができない場合、ヌルポインタ (NULL) を返します。

参照 [「vec_calloc」\(p270\)](#)、[「malloc」\(p260\)](#)、[「realloc」\(p265\)](#)

例 26.6 `calloc()` の使用例

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    static char s[] = "Metrowerks compilers";
    char *sptr1, *sptr2, *sptr3;

    // メモリを 3 つの異なる方法で割り当てる。
    // 1 番目: 初期化しないメモリブロックを
    // 30 バイト割り当てる。
    sptr1 = (char *) malloc(30);
    strcpy(sptr1, s);
    printf("Address of sptr1: %p\n", sptr1);

    // 2 番目: 初期化しないメモリを 20 バイト割り当てる。
    sptr2 = (char *) malloc(20);
    printf("sptr2 before reallocation: %p\n", sptr2);
    strcpy(sptr2, s);
    // 10 バイト余分に再割り当てを行う。
    // (全部で 30 バイト)
    //
    // sptr2 によって指されるメモリブロックは、realloc() が
    // 呼ばれた後も連続していることに注意。
    sptr2 = (char *) realloc(sptr2, 30);
```

```
printf("sptr2 after reallocation: %p\n", sptr2);

// 3 番目：初期化されたメモリを 30 バイト割り当てる。
sptr3 = (char *) calloc(strlen(s), sizeof(char));
strcpy(sptr3, s);
printf("Address of sptr3: %p\n", sptr3);

puts(sptr1);
puts(sptr2);
puts(sptr3);

// 割り当てられたメモリをヒープに解放する。
free(sptr1);
free(sptr2);
free(sptr3);

return 0;
}
```

Output:
Address of sptr1: 5e5432
sptr2 before reallocation: 5e5452
sptr2 after reallocation: 5e5468
Address of sptr3: 5e5488
Metrowerks compilers
Metrowerks compilers
Metrowerks compilers

div

整数の商と剰余を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <stdlib.h>

引数 この関数の引数です。

numer	int	numerator
denom	int	denominator

注意点 div_t 型は、stdlib.h で typedef struct { int quot,rem; } div_t; と定義されます。

戻り値 `div()` は `denom` を `numer` で割り、`div_t` 型の商と剰余を返します。

参照 [「fmod」\(p94\)](#)、[「ldiv」\(p260\)](#)、[「div_t」\(p47\)](#)

例 26.7 `div()` の使用例

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    div_t result;
    ldiv_t lresult;

    int d = 10, n = 103;
    long int ld = 1000L, ln = 1000005L;

    result = div(n, d);
    lresult = ldiv(ln, ld);

    printf("%d / %d has a quotient of %d\n",
           n, d, result.quot);
    printf("and a remainder of %d\n", result.rem);
    printf("%ld / %ld has a quotient of %ld\n",
           ln, ld, lresult.quot);
    printf("and a remainder of %ld\n", lresult.rem);

    return 0;
}
```

Output:

```
103 / 10 has a quotient of 10
and a remainder of 3
1000005 / 1000 has a quotient of 1000
and a remainder of 5
```

exit

プログラムを正常終了させます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdlib.h>
void exit(int status);
```


引数 この関数の引数です。

`status` `int` `exit` エラーの値

注意点 `exit()` 関数は、`atexit()` で登録された全ての関数を登録された逆の順番で実行します。そして、バッファをフラッシュし、オープンされたストリームの全てをクローズします。そのとき Toolbox のシステムコール `ExitToShell()` を呼び出します。

戻り値 `exit()` は、オペレーティングシステムへ値を返しません。引数 `status` は、ANSI C の標準ライブラリ仕様に準拠する値を保持します。

参照 [「abort」\(p244 \)](#)、[「atexit」\(p246 \)](#)

例 26.8 `exit()` の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *f;
    int count;

    // 出力用の新しいファイルを生成し、失敗の場合は終了する
    if (( f = fopen("foofoo", "w")) == NULL) {
        printf("Can't create file.\n");
        exit(1);
    }

    // 0 から 9 の値を出力
    for (count = 0; count < 10; count++)
        fprintf(f, "%5d", count);

    // ファイルをクローズ
    fclose(f);

    return 0;
}
```

`free`

以前に割り当てられたメモリをヒープに解放します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`
`void free(void *ptr);`

引数 この関数の引数です。

`ptr` `void *` 割り当てられたメモリへのポインタ

注意点 `free()` 関数は、`ptr` によって指される過去に割り当てられたメモリブロックを、ヒープに解放します。引数 `ptr` では、メモリ割り当て関数 `calloc()`、`malloc()`、`realloc()` によって返されたアドレスが保持されています。`ptr` で指されたメモリブロックは、一旦解放されると有効ではなくなります。変数 `ptr` は、メモリ割り当て関数からある値が割り当てられるまで、再びメモリを参照するのに使用すべきではありません。

参照 [「vec_free」\(p271\)](#) [「calloc」\(p253\)](#) [「malloc」\(p260\)](#) [「realloc」\(p265\)](#)

例 26.9 `free()` 使用例

[「calloc\(\) の使用例」\(p254\)](#) を参照してください。

`getenv`

環境リストのアクセス

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`
`char *getenv(const char *name);`

引数 この関数の引数です。

`name` `const char *` 環境リストのバッファ

注意点 Mac OS システム上では、`getenv()` は常にヌルポインタ (NULL) を返す空の関数です。これは、CodeWarrior の `stdlib.h` ヘッダに含まれており ANSI C の標準ライブラリ仕様に準拠しています。

戻り値 Mac OS システム上では、`getenv()` はヌルポインタ (NULL) を返します。Windows 上では `getenv()` は失敗したときにゼロ、または環境変数を返します。

参照 [「system」\(p270\)](#)

例 26.10 getenv() の使用例

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char *value;
    char *var = "path";

    if( (value = getenv(var)) == NULL)
    { printf("%s is not a environmental variable", var);}
    else
    { printf("%s = %s \n", var, value);}

    return 0;
}

Result:
path = c:\program files\metrowerks\codewarrior;c:\WINNT\system32
```

labs

long integer 型の絶対値を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdlib.h>
long int labs(long int j);
```

引数 この関数の引数です。

j long int 計算される変数

戻り値 labs() は、その引数の long int 型としての絶対値を返します。

参照 [「fabs」\(p92\)](#)、[「abs」\(p245\)](#)

例 26.11 labs() の使用例

[「abs\(\) の使用例」\(p246\)](#) を参照してください。

ldiv

long の整数型の商と剰余を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`
`ldiv_t ldiv(long int numer, long int denom);`

引数 この関数の引数です。

numer	long int	numerator
denom	long int	denominator

注意点 `ldiv_t` 型は `stdlib.h` で、

```
typedef struct {  
    long int quot, rem;  
} ldiv_t;
```


と定義されます。

戻り値 `ldiv()` は、`denom` を `numer` で割り、`ldiv_t` 型としての商と剰余を返します。

参照 [「fmod」\(p94\)](#)、[「div」\(p255\)](#)、[「ldiv_t」\(p47\)](#)

例 26.12 ldiv() の使用例

[「div\(\) の使用例」\(p256\)](#) を参照してください。

malloc

ヒープメモリの 1 ブロックを割り当てます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`
`void *malloc(size_t size);`

引数 この関数の引数です。

size	size_t	割り当てのバイトサイズ
------	--------	-------------

注意点 malloc() 関数は、size バイトの連続したヒープメモリの 1 ブロックを割り当てます。

戻り値 malloc() は、成功した場合は割り当てられたブロックの最初のバイトへのポインタを返し、失敗した場合はヌルポインタを返します。

参照 [「vec_malloc」\(p271\)](#)、[「calloc」\(p253\)](#)、[「free」\(p257\)](#)、[「realloc」\(p265\)](#)

例 26.13 malloc() の使用例

[「calloc\(\) の使用例」\(p254\)](#) を参照してください。

mblen

マルチバイトの文字の長さを計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdlib.h>
int mblen(const char *s, size_t n);
```

引数 この関数の引数です。

s	const char *	計算するマルチバイト配列
n	size_t	最大サイズ

注意点 mblen() 関数は、s によって指し示されたマルチバイトの文字の長さを返します。これは最大 n 文字分について調べます。

CodeWarrior C の実装においては、C の場合のみサポートし、mbtowc(NULL, s, n) の値を返します。

戻り値 mblen() は、mbtowc(NULL, s, n) の値を返します。

参照 [「地域設定情報の仕様」\(p73\)](#)、[「mbtowc」\(p262\)](#)

mbstowcs

マルチバイトの文字配列を wchar_t 型の配列に変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`
`size_t mbstowcs(wchar_t *pwcs,`
`const char *s, size_t n);`

引数 この関数の引数です。

<code>pwcs</code>	<code>wchar_t*</code>	拡張文字のデスティネーション
<code>s</code>	<code>const char *s</code>	変換する文字列
<code>n</code>	<code>size_t</code>	変換する最大拡張文字

注意点 `mbstowcs()` 関数は、マルチバイトの文字配列を `wchar_t` 型の文字配列に変換します。
`wchar_t` 型は、`stddef.h` で定義されています。

CodeWarrior C での `mbstowcs()` 関数は、何も変換しません。これは、`s` で指された配列から `pwcs` で指された配列へ最大 `n` バイトのコピーを行います。関数はヌルポインタに先に到達した場合、そこで終了します。

戻り値 `mbstowcs()` は、`s` から `pwcs` へコピーされたバイト数を返します。

参照 [「地域設定情報の仕様」\(p73 \)](#)、[「wcstombs」\(p272 \)](#)

mbtowc

マルチバイトの文字を `wchar_t` 型に変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
-------------	-------------	-----------------	---------------	----------------	--------------	--

プロトタイプ `#include <stdlib.h>`
`int mbtowc(wchar_t *pwc,`
`const char *s, size_t n);`

引数 この関数の引数です。

<code>pwc</code>	<code>wchar_t*</code>	拡張文字のデスティネーション
<code>s</code>	<code>const char *s</code>	変換する文字列
<code>n</code>	<code>size_t</code>	変換する最大拡張文字

注意点 `mbtowc()` 関数は、`s` で指されたマルチバイトの文字を `pwc` で指された `wchar_t` 型の文字へ変換します。関数は、最大 `n` バイトの変換を行います。

CodeWarrior C の `mbtowc()` 関数では、何も変換しません。`s` に関する最初の文字を `pwc` の最初の文字にコピーします。

戻り値 `mbtowc()` は、`n` がゼロで `s` がヌルポインタでない場合、-1 を返します。

`mbtowc()` は、`s` がヌルポインタか、`s` がヌル文字 (`\0`) を指した場合、0 を返します。

`mbtowc()` は、`s` がヌルポインタでなく、ヌル文字を指していない場合、1 を返します。

参照 [「地域設定情報の仕様」\(p73 \)](#)、[「mblen」\(p261 \)](#)、[「wctomb」\(p273 \)](#)

qsort

配列をソートします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`
`void qsort(void *base, size_t nmemb,`
 `size_t size,`
 `int (*compare) (const void *, const void *))`

引数 この関数の引数です。

<code>base</code>	<code>void *</code>	保存される配列へのポインタ
<code>nmemb</code>	<code>size_t</code>	要素の数
<code>size</code>	<code>size_t</code>	要素のサイズ
<code>compare</code>	<code>void *</code>	比較関数へのポインタ

注意点 `qsort()` 関数は、クイックソートアルゴリズムを使って配列をソートします。これは、配列を移動させることなくソートを行います。配列は `qsort()` を呼ぶ前と同じメモリが使用されます。

引数 `base` は、ソートされる配列の基底アドレスへポインタです。

引数 `nmemb` は、ソートのための配列要素の数を指定します。

引数 `size` は、配列要素のサイズを指定します。

引数 `compare` は、プログラマが提供する比較 (`compare`) 関数へのポインタです。関数は、異なる配列の要素への 2 つのポインタを持ち、それらをキーに基づいて比較します。2 つの要素が等しい場合、`compare` はゼロを返します。`compare` 関数は、最初の要素が 2 番目の要素より小さければ負の数を返します。同様に、最初の要素が 2 番目より大きい場合、正の数を返します。

参照 [「bsearch」\(p250 \)](#)

例 26.14 `qsort()` の使用例

[「bsearch の使用例」\(p251 \)](#) を参照してください。

rand

疑似乱数を生成します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`
`int rand(void);`

引数 なし。

注意点 `rand()` 関数を連続的に呼ぶと、0 から `RAND_MAX` の疑似乱数の整数値を連続的に生成してその値を返します。`RAND_MAX` マクロは、`stdlib.h` で定義されています。

`srand()` を使って種 (seeding) を使った乱数生成は、`rand()` を使って生成される乱数の系列とは異なります。

戻り値 `rand()` は、疑似乱数である 0 と `RAND_MAX` の間の整数値を返します。

参照 [「srand」\(p266\)](#)

例 26.15 rand() の使用例

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    int i;
    unsigned int seed;

    for (seed = 1; seed <= 5; seed++) {
        srand(seed);
        printf("First five random number for seed %d:\n",
               seed);
        for (i = 0; i < 5; i++)
            printf("%10d", rand());
        printf("\n\n"); // 行を終わる
    }

    return 0;
}
```

Output:
First five random number for seed 1:
16838 5758 10113 17515 31051


```

First five random number for seed 2:
    908      22817      10239      12914      25837

First five random number for seed 3:
   17747      7107      10365      8312      20622

First five random number for seed 4:
    1817      24166      10491      3711      15407

First five random number for seed 5:
   18655      8457      10616      31877      10193

```

realloc

割り当てられたヒープメモリの 1 ブロックのサイズを変更します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <stdlib.h>`
`void *realloc(void *ptr, size_t size);`

引数 この関数の引数です。

<code>ptr</code>	<code>void *</code>	割り当てられたメモリブロックへのポインタ
<code>size</code>	<code>size_t</code>	割り当てるメモリのサイズ

注意点 `realloc()` 関数は、`ptr` で指されたメモリブロックのサイズを `size` バイトに変更します。引数 `size` は、現在 `ptr` で指されているブロックのサイズよりも小さい、または大きい値であってもよいです。引数 `ptr` は、メモリ割り当て関数 `calloc()` と `malloc()` によって割り当てられた値です。

サイズが 0 の場合、`ptr` によって指されたメモリブロックは解放されます。ヌルポインタである場合、`realloc()` は `size` バイトを割り当てます。

新しいメモリブロック中での古いメモリブロック内容は、新しいブロックの方が古いものより大きい場合、保持されます。新しいブロックが小さい場合、増えたバイト数分古いブロックの最後から削除されます。

戻り値 `realloc()` は、`size` が 0 以上で成功した場合、新しいブロックへのポインタを返します。`size` が 0 あるいは失敗した場合、ヌルポインタを返します。

参照 [「vec_realloc」\(p271\)](#)、[「calloc」\(p253\)](#)、[「free」\(p257\)](#)、[「malloc」\(p260\)](#)

例 26.16 realloc() の使用例

[「calloc\(\) の使用例」\(p254\)](#) を参照してください。

srand

疑似乱数発生器の種を設定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdlib.h>
void srand(unsigned int seed);
```

引数 この関数の引数です。

seed unsigned int 疑似乱数発生器の種の値

注意点 srand() 関数は、疑似乱数発生器の種を設定します。それぞれの種の値は、利用された場合に同じ乱数の系列を生成します。

参照 [「rand」\(p264\)](#)

例 26.17 labs() の使用例

[「rand\(\) の使用例」\(p264\)](#) を参照してください。

strtod

文字を数値へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdlib.h>
double strtod( const char *nptr,
               char **endptr);
```

引数 この関数の引数です。

nptr const char * 変換する Null 終了した配列
endptr char ** 変換不可能な nptry の位置へのポインタ

注意点 `strtod()` は、`nptr` で指された文字配列から `double` 型の浮動小数点の値へ変換します。文字配列は、10 進表記 (例えば 103.578) あるいは指数表現 (`[-]b.aaae ± Eee`) のいずれかが可能です。

引数 `endptr` がヌルポインタでない場合、`nptr` で指される文字配列内の位置のポインタとして割り当てられます。この位置は最初の文字を示し、関数のそれぞれの型へは変換できません。

この関数は最初の空白をスキップします。

この関数は、変換エラーが生じると大域変数 `errno` に `ERANGE` を設定します。

戻り値 `strtod()` は、`double` 型の浮動小数点の値を返します。`nptr` が表現可能な `double` 型の値に変換できない場合、`strtod()` は `math.h` で定義された `HUGE_VAL` を返し、`errno` に `ERANGE` を設定します。

参照 [「strtol」\(p268\)](#)、[「strtoul」\(p269\)](#)、[「errno」\(p49\)](#)、[「整数型の限界」\(p71\)](#)、[「math.h の概要」\(p79\)](#)、[「scanf」\(p224\)](#)

例 26.18 `strtod()`、`strtol()`、`strtoul()` の使用例

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    double f;
    long int i;
    unsigned long int j;
    static char si[] = "4733!", sf[] = "103.749?";
    static char sb[] = "0x10*";
    char *endptr;

    f = strtod(sf, &endptr);
    printf("%f %c\n", f, *endptr);

    i = strtol(si, &endptr, 10);
    printf("%ld %c\n", i, *endptr);

    i = strtol(si, &endptr, 8);
    printf("%ld %c\n", i, *endptr);

    j = strtoul(sb, &endptr, 0);
    printf("%ld %c\n", j, *endptr);

    j = strtoul(sb, &endptr, 10);
    printf("%ld %c\n", j, *endptr);
}
```

```
    return 0;
}
```

Output:
103.749000 ?
4733 !
2523 !
16 *
0 x

strtol

文字配列から数値へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `long int strtol(const char *nptr,
char **endptr, int base);`

引数 この関数の引数です。

<code>nptr</code>	<code>const char *</code>	変換する Null 終了した配列
<code>endptr</code>	<code>char **</code>	変換不可能な <code>nptr</code> の位置へのポインタ
<code>base</code>	<code>int</code>	2 から 36 間での基底数値

注意点 `strtol()` 関数は、`nptr` で指された文字配列から基底付きの `long int` 型の整数値へ変換します。数値の文字列の前に付く、プラスあるいはマイナス符号 (+ または -) は、オプションです。

`strtol()` と `strtoul()` の引数 `base` では、変換のための基底を指定します。これは 2 から 36 の間、または 0 の値を持つ必要があります。`base` が 0 の場合、`strtol()` と `strtoul()` はその書式に基づいて文字配列を変換します。'0' で始まる文字配列は 8 進数と仮定され、'0x' や '0X' で始まる数は 16 進数と仮定されます。他の数字の文字列は 10 進数と仮定されます。

引数 `endptr` がヌルポインタでない場合、`nptr` で指される文字配列内の位置のポインタとして割り当てられます。この位置は最初の文字を示し、関数のそれぞれの型へは変換できません。

この関数は最初の空白をスキップします。

この関数は、変換エラーが発生すると大域変数 `errno` に `ERANGE` を設定します。

戻り値 `strtol()` は、`long int` 型の整数値を返します。変換された値が `LONG_MIN` より小さい場合、`strtol()` は `LONG_MIN` を返し、`errno` に `ERANGE` を設定します。変換された値が

LONG_MAX より大きい場合、`strtol()` は LONG_MAX を返し `errno` に ERANGE を設定します。LONG_MIN と LONG_MAX マクロは `limits.h` で定義されます。

参照 [「strtod」\(p266\)](#)、[「strtoul」\(p269\)](#)、[「errno」\(p49\)](#)、[「整数型の限界」\(p71\)](#)、[「math.h の概要」\(p79\)](#)、[「scanf」\(p224\)](#)

strtoul

文字配列を数値へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `unsigned long int strtoul(const char *nptr, char **endptr, int base);`

引数 この関数の引数です。

<code>nptr</code>	<code>const char *</code>	変換する Null 終了した配列
<code>endptr</code>	<code>char **</code>	変換不可能な <code>nptr</code> の位置へのポインタ
<code>base</code>	<code>int</code>	2 から 36 までの基底数値

注意点 `strtoul()` 関数は、`nptr` で指された文字配列から基底付きの `long int` 型の整数値へ変換します。前に付加されたプラスあるいはマイナス符号 (+ または -) は無視されます。

`strtol()` と `strtoul()` の引数 `base` では、変換のための基底を指定します。これは 2 から 36 の間、または 0 の値を持つ必要があります。`base` が 0 の場合、`strtol()` と `strtoul()` はその書式に基づいて文字配列を変換します。'0' で始まる文字配列は、8 進数と仮定され、'0x' や '0X' で始まる数は 16 進数となります。他の数字の文字列は 10 進数と仮定されます。

引数 `endptr` がヌルポインタでない場合、`nptr` で指される文字配列内の位置のポインタとして割り当てられます。この位置目印は最初の文字を示し、関数のそれぞれの型へは変換できません。

この関数は最初の空白をスキップします。

この関数は、変換エラーが生じると大域変数 `errno` に ERANGE を設定します。

戻り値 `strtoul()` は、`unsigned long int` 型の整数値を返します。変換された値が ULONG_MAX より大きい場合、`strtol()` は ULONG_MAX を返し `errno` に ERANGE を設定します。ULONG_MAX マクロは `limits.h` で定義されます。

参照 [「strtod」\(p266\)](#)、[「strtol」\(p268\)](#)、[「errno」\(p49\)](#)、[「整数型の限界」\(p71\)](#)、[「math.h の概要」\(p79\)](#)、[「scanf」\(p224\)](#)

system

環境リストの割付を行います。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdlib.h>
int system(const char *string);
```

警告! system() 関数は、ANSI C の標準ライブラリ仕様に準拠する CodeWarrior の stdlib.h に含まれる空の関数です。

引数 この関数の引数です。

string const char * OS システムコマンド

戻り値 system() は常時 0 を返します。

参照 [「getenv」\(p258\)](#)

vec_alloc

16 ビットアライメント上のメモリを消去、および割り当てます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	Altivec
------	------	----------	--------	---------	-------	---------

プロトタイプ

```
void *vec_alloc(size_t nmemb, size_t size);
```

引数 この関数の引数です。

nmemb size_t 要素の数
size size_t 要素のサイズ

注意 vec_alloc() 関数は nmemb 要素のサイズに対して連続したスペースを割り当てます。スペースはゼロで初期化されます。

戻り値 vec_alloc() は、割り当てたメモリの第 1 バイトへのポインタを返します。スペースを割り当てられないとき、vec_alloc() はヌルポインタ (NULL) を返します。

参照 [「calloc」\(p253\)](#)、[「vec_free」\(p271\)](#)、[「vec_malloc」\(p271\)](#)、[「vec_realloc」\(p271\)](#)

vec_free

vec_malloc、vec_calloc、vec_realloc によって割り当てられたメモリを解放します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	Altivec
------	------	----------	--------	---------	-------	---------

プロトタイプ void vec_free(void *ptr);

引数 この関数の引数です。

ptr void * 割り当てられたメモリへのポインタ

注意 vec_free() 関数は、ptr 引数が指す、以前に割り当てられたメモリブロックをヒープへ解放します。ptr 引数は、メモリ割り当て関数 (vec_malloc()、vec_calloc()、vec_realloc()) が返したアドレスを持ちます。ptr が指すメモリブロックが解放されると、ptr はすぐに無効になります。メモリ割り当て関数からの値が ptr に代入されるまでは、ptr 変数を再度メモリを参照するためには使用しないでください。

戻り値 なし。

参照 [「free」\(p257\)](#)、[「vec_calloc」\(p270\)](#)、[「vec_malloc」\(p271\)](#)、[「vec_realloc」\(p271\)](#)

vec_malloc

16 ビットアライメント上にメモリを割り当てます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	Altivec
------	------	----------	--------	---------	-------	---------

プロトタイプ void *vec_malloc(size_t size);

引数 この関数の引数です。

size size_t 割り当てるバイトのサイズ

注意 vec_malloc() 関数は、size バイト量の連続したヒープメモリブロックを割り当てます。

戻り値 成功した場合、vec_malloc() は割り当てられたブロックの第一バイトへのポインタを返します。失敗した場合、ヌルポインタを返します。

参照 [「malloc」\(p260\)](#)、[「vec_free」\(p271\)](#)、[「vec_calloc」\(p270\)](#)、[「vec_realloc」\(p271\)](#)

vec_realloc

16 ビットアライメント上にメモリを再度割り当てます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	Altivec
------	------	----------	--------	---------	-------	---------

プロトタイプ `void *vec_realloc(void * ptr, size_t size);`

引数 この関数の引数です。

<code>ptr</code>	<code>void *</code>	割り当てられたメモリブロックへのポインタ
<code>size</code>	<code>size_t</code>	再度割り当てるメモリのサイズ

戻り値 `size` が 0 よりも大きく、再割り当てが成功した場合、`vec_realloc()` は新しいブロックへのポインタを返します。失敗した場合、または `size` が 0 の場合、ヌルポインタを返します。

参照 [「realloc」\(p265\)](#)、[「vec_free」\(p271\)](#)、[「vec_calloc」\(p270\)](#)、[「vec_malloc」\(p271\)](#)

wcstombs

`wchar_t` 型の文字配列をマルチバイトの文字配列に変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdlib.h>
size_t wcstombs(char *s, const
                wchar_t *pwcs, size_t n);
```

引数 この関数の引数です。

<code>s</code>	<code>char *</code>	マルチバイト文字列バッファ
<code>pwcs</code>	<code>const wchar_t *</code>	変換する拡張文字列へのポインタ
<code>n</code>	<code>size_t</code>	変換する最大長

注意点 `wcstombs()` 関数は、`wchar_t` 型の文字配列からマルチバイトの文字配列へ変換します。`wchar_t` 型は `stddef.h` で定義されます。

CodeWarrior C の `wcstombs()` は何も変換しません。これは、`pwcs` で指された配列から `s` で指された配列へ最大 `n` バイトのコピーを行います。関数は、ヌルポインタに先に到達した場合、そこで終了します。

戻り値 `wcstombs()` は、`pwcs` から `s` へコピーされたバイト数を返します。

参照 [「地域設定情報の仕様」\(p73\)](#)、[「mbstowcs」\(p261\)](#)

wctomb

wchar_t 型からマルチバイトの文字に変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <stdlib.h>
int wctomb(char *s, wchar_t wchar);
```

引数 この関数の引数です。

s	char *	マルチバイト文字列バッファ
wchar	wchar_t	変換する拡張文字

注意点 wctomb() 関数は、wchar_t 型の文字からマルチバイトの文字へ変換します。

CodeWarrior C の wctomb() は何も変換しません。これは、wchar を s で指された文字へ割り当てます。

戻り値 s がヌルではない場合、wctomb() は 1 を返します。それ以外の場合、0 を返します。

参照 [「地域設定情報の仕様」\(p73\)](#)、[「mbtowc」\(p262\)](#)

第 27 章 string.h

string.h ヘッダファイルは、文字配列やより大きい項目の配列を比較、複写、結合、検索する関数を提供します。

string.h の概要

string.h で使われる関数の命名規約では、関数の操作するデータ構造の型でその名前を決定します。

str が前に付く関数は、ヌル文字 (\0) で終了する文字配列に作用します。str に関する関数を以下に示します。

[「strcasecmp」\(p281 \)](#): 大文字、小文字の違いを無視して文字列の比較する

[「strcat」\(p281 \)](#): 文字列を結合する

[「strchr」\(p282 \)](#): 文字で検索する

[「strcmp」\(p283 \)](#): 文字列を比較する

[「strcpy」\(p285 \)](#): 文字列の複写する

[「strcoll」\(p284 \)](#): 文字列を辞書式順番で比較する

[「strcspn」\(p286 \)](#): 文字列中の部分文字列を検索する

[「strdup」\(p287 \)](#): (Windows_strdup) 文字列を複写する

[「strerror」\(p288 \)](#): 変数 errno からエラーメッセージを検索する

[「strlen」\(p289 \)](#): 文字列の長さを返す

[「strpbrk」\(p294 \)](#): 文字列中の 1 文字をもう一方の文字列から探す

[「strrchr」\(p295 \)](#): 1 文字を文字列から探す

[「_strrev」\(p296 \)](#): 文字列を反転させる

[「strspn」\(p296 \)](#): 文字列にない 1 文字をもう一方の文字列から探す

[「strstr」\(p297 \)](#): 文字列を文字列から検索する

[「strtok」\(p298 \)](#): 次のトークンあるいは部分文字列を検索する

[「strxfrm」\(p300 \)](#): 文字列を地域設定コードに変換する

[「strupr」\(p301 \)](#): 文字列を大文字に変換する

strn が名前の前に付加された関数はその引数で指定された長さの文字配列上で作用します。strn 関数を以下に示します。

[「strncasecmp」\(p290 \)](#): 指定された長さで文字列の大文字、小文字の違いを考慮して比較する

[「strncasecmp」\(p290 \)](#): 指定された長さを用いて文字列を結合する

[「strncmp」\(p291 \)](#): 指定された長さを用いて文字列を比較する

[「strncpy」\(p292 \)](#): 文字列の長さを用いて文字列を複写する

mem が名前の前に付けられた関数は、項目の配列あるいはメモリの連続したブロック上で作用します。配列やメモリのブロックのサイズは、関数の引数として指定されます。mem 関数を以下に示します。

[「memchr」\(p276 \)](#): 1 文字をメモリブロックで探す

[「memcmp」\(p278 \)](#): メモリブロックを比較する

[「memcpy」\(p279 \)](#): メモリブロックをコピーする

[「memmove」\(p279 \)](#): メモリブロックを移動する

[「memset」\(p280 \)](#): メモリブロックに値を設定する

'stri' が名前の前に付けられた関数は大文字、小文字の違いを無視した文字列上で作用します。

[「_stricmp」\(p288 \)](#): 大文字、小文字の違いを無視して比較する

[「_strnicmp」\(p293 \)](#): 指定された長さで、大文字、小文字の違いを無視して文字列を比較する

memchr

1 文字を検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <string.h>
void *memchr(const void *s, int c, size_t n);
```

引数 この関数の引数です。

s	const void *	検索するメモリ
c	int	検索する char
n	size_t	検索する最大長

注意点 memchr () 関数は、s によって指されるメモリ領域の最初の n 文字中の最初の c を探します。

戻り値 memchr () は、見つけた文字を指すポインタを返します。c が見つからなければヌルポインタ (NULL) を返します。

参照 [「strchr」\(p282\)](#)、[「strchr」\(p295\)](#)

例 27.1 memchr() の使用例

```
#include <string.h>
#include <stdio.h>

#define ARRAYSIZE 100

int main(void)
{
    // この例では、s1 は s2 と同じ長さでなければならない！
    static char s1[ARRAYSIZE] = "laugh* giggle 231!";
    static char s2[ARRAYSIZE] = "grunt sigh# snort!";
    char dest[ARRAYSIZE];
    char *strpstr;
    int len1, len2, lendest;

    // memset() を使って目的の文字列をクリアする。
    memset( (char *)dest, '\0', ARRAYSIZE);

    // 文字列の長さは、mem 関数に必要。
    // 終了の '\0' 文字が含まれるため 1 を加える。
    len1 = strlen(s1) + 1;
    len2 = strlen(s2) + 1;
    lendest = strlen(dest) + 1;

    printf(" s1=%s\n s2=%s\n dest=%s\n\n", s1, s2, dest);

    if (memcmp( (char *)s1, (char *)s2, len1) > 0)
        memcpy( (char *)dest, (char *)s1, len1);
    else
        memcpy( (char *)dest, (char *)s2, len2);

    printf(" s1=%s\n s2=%s\n dest=%s\n\n", s1, s2, dest);

    // memchr() と memmove() を使って s1 をそれ自身にコピー。
    strpstr = (char *)memchr( (char *)s1, '*', len1);
    memmove( (char *)strpstr, (char *)s1, len1);

    printf(" s1=%s\n s2=%s\n dest=%s\n\n", s1, s2, dest);

    return 0;
}
```

Output:
s1=laugh* giggle 231!
s2=grunt sigh# snort!
dest=

```
s1=laugh* giggle 231!  
s2=grunt sigh# snort!  
dest=laugh* giggle 231!  
  
s1=laughlaugh* giggle 231!  
s2=grunt sigh# snort!  
dest=laugh* giggle 231!
```

memcmp

2つのメモリブロックを比較します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`int memcmp(const void *s1,`
`const void *s2, size_t n);`

引数 この関数の引数です。

s1	const void *	比較するメモリ
s2	const void *	比較メモリ
n	size_t	比較する最大長

注意点 `memcmp()` 関数は、s1 と s2 の最初の n 文字を一度に 1 文字ずつ比較します。

戻り値 `memcmp()` は、s1 と s2 によって指された全ての n 文字が同じである場合、ゼロを返します。

`memcmp()` は、s1 で指された最初のマッチしない文字が s2 で指された文字より小さい場合、負の値を返します。

`memcmp()` は、s1 で指された最初のマッチしない文字が、s2 で指された文字より大きい場合、正の値を返します。

参照 [「strcmp」\(p283\)](#)、[「strncmp」\(p291\)](#)

例 27.2 memcmp() の使用例

[「memchr\(\) の使用例」\(p277\)](#) を参照してください。

memcpy

連続したメモリブロックをコピーします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <string.h>
void *memcpy(const void *dest,
             const void *source, size_t n);
```

引数 この関数の引数です。

dest	void *	デスティネーションメモリ
source	const void *	コピーするソース
n	size_t	コピーする最大長

注意点 memcpy() 関数は、source によって指された項目から dest によって指された項目へ最初から n 文字分コピーします。dest と source で指された領域が重複した場合、memcpy() の動作は定義されていません。memmove() 関数は、重複したメモリブロックに対して信頼性のあるコピーを行います。

戻り値 memcpy() は dest の値を返します。

参照 [「memmove」\(p279\)](#)、[「strcpy」\(p285\)](#)、[「strncpy」\(p292\)](#)

例 27.3 memcpy() の使用例

[「memchr\(\) の使用例」\(p277\)](#) を参照してください。

memmove

重複した連続のメモリブロックをコピーします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <string.h>
void *memmove(void *dest,
              const void *source, size_t n);
```

引数 この関数の引数です。

dest	void *	デスティネーションメモリ
source	const void *	移動するソース
n	size_t	移動する最大長

注意点 memmove() 関数は、source によって指された項目の最初の n 文字分を、dest によって指された項目へコピーします。

memcpy() と異なり、memmove() 関数は重複したメモリブロックを安全にコピーします。

戻り値 memmove() は、dest の値を返します。

参照 [「memcpy」\(p279\)](#)、[「memset」\(p280\)](#)、[「strcpy」\(p285\)](#)、[「strncpy」\(p292\)](#)

例 27.4 memmove() の使用例

[「memchr」の使用例」\(p277\)](#) を参照してください。

memset

メモリブロックの内容をクリアします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <string.h>
void *memset(void *dest, int c, size_t n);
```

引数 この関数の引数です。

dest	void *	デスティネーションメモリ
c	int	セットする char
n	size_t	セットする最大長

注意点 memset() 関数は、dest によって指される項目の最初の n 文字に c を割り当てます。

戻り値 memset() は、dest の値を返します。

例 27.5 memset() の使用例

[「memchr」の使用例」\(p277\)](#) を参照してください。

strcasecmp

大文字、小文字の違いを考慮せずに文字列の比較を行う関数です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <string.h>
int strcasecmp (
    const char *str1,
    const char *str2);
```

引数 この関数の引数を以下に示します。

str1	const char *	比較される文字列
str2	const char *	比較文字列

戻り値 Strcasecmp は、str1 が str2 より大きい場合、ゼロより大きな値を返し、str2 が str1 より大きい場合、ゼロより小さい値を返します。両方が等しい場合はゼロを返します。

参照 [「strncasecmp」\(p290 \)](#)

strcat

2 つの文字配列を連結します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <string.h>
char *strcat(char *dest, const char *source);
```

引数 この関数の引数です。

dest	char *	デスティネーション文字列
source	const char *	追加するソース

注意点 strcat() 関数は、source によって指された文字配列のコピーを dest によって指された文字配列の終わりに追加します。引数 dest と source は、ヌルで終了する文字配列を指し示さなければなりません。strcat() は、ヌルで終了する文字配列を生成します。

戻り値 strcat() は、dest の値を返します

参照 [「strncasecmp」\(p290 \)](#)

例 27.6 strcat() の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    char s1[100] = "The quick brown fox ";
    static char s2[] = "jumped over the lazy dog.";

    strcat(s1, s2);
    puts(s1);

    return 0;
}
```

Output:
The quick brown fox jumped over the lazy dog.

strchr

1 文字を検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`char *strchr(const char *s, int c);`

引数 この関数の引数です。

s	const char *	検索する文字列
c	int	検索する char

注意点 `strchr()` 関数は、s によって指された文字配列中の最初の文字 c を検索します。引数 s は、ヌルで終了する文字配列を指さなければなりません。

戻り値 `strchr()` は、成功した場合、検索した文字へのポインタを返します。失敗した場合、`strchr()` はヌルポインタ (NULL) を返します。

参照 [「memchr」\(p276 \)](#)、[「strchr」\(p295 \)](#)

例 27.7 strchr() の使用例

```
#include <string.h>
#include <stdio.h>
```

```
int main(void)
{
    static char s[] = "tree * tomato eggplant garlic";
    char *strpstr;

    strpstr = strchr(s, '*');
    puts(strpstr);

    return 0;
}
```

Output:
* tomato eggplant garlic

strcmp

2 つの文字配列を比較します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`int strcmp(const char *s1, const char *s2);`

引数 この関数の引数です。

s1	const char *	比較する文字列
s2	const char *	比較文字列

注意点 `strcmp()` 関数は、s1 によって指された文字配列と s2 によって指された文字配列の比較を行います。s1 と s2 の両方ともヌルで終了する文字配列を指さなければなりません。

戻り値 `strcmp()` は、s1 と s2 が同じである場合、ゼロを返し、s1 が s2 より小さければ負の値、s1 が s2 より大きければ正の値を返します。

参照 [「memcmp」\(p278\)](#)、[「strcoll」\(p284\)](#)、[「strncmp」\(p291\)](#)

例 27.8 strcmp() の使用例

```
#include <string.h>
#include <stdio.h>

int main (void)
{
```

```
static char s1[] = "butter", s2[] = "olive oil";
char dest[20];

if (strcmp(s1, s2) < 0)
    strcpy(dest, s2);
else
    strcpy(dest, s1);

printf(" s1=%s\n s2=%s\n dest=%s\n", s1, s2, dest);

return 0;
}
```

Output:
s1=butter
s2=olive oil
dest=olive oil

strcoll

2 つの地域設定（コード）に対応した文字配列を比較します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`int strcoll(const char *s1, const char *s2);`

引数 この関数の引数です。

s1	const char *	比較する文字列
s2	const char *	比較文字列

注意点 `strcoll()` 関数は、`locale.h` ヘッダファイルによる照合シーケンスセット（collating sequence set）に基づいて 2 つの文字配列を比較します。

CodeWarrior C の `strcoll()` では、`strcmp()` を用いて 2 つの文字配列を比較します。これは文字列のライブラリの中に含まれており、ANSI C の標準ライブラリ仕様に準拠しています。

戻り値 `s1` と `s2` が等しい場合、`strcoll()` はゼロを返し、`s2` より `s1` の方が小さい場合、負の値、`s1` が `s2` より大きい場合、正の値を返します。

参照 [「地域設定情報の仕様」\(p73\)](#)、[「memcmp」\(p278\)](#)、[「strcmp」\(p283\)](#)、[「strncmp」\(p291\)](#)

例 27.9 strcoll() の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    static char s1[] = "aardvark", s2[] = "xylophone";
    int result;

    result = strcoll(s1, s2);

    if (result < 1)
        printf("%s is less than %s\n", s1, s2);
    else
        printf("%s is equal or greater than %s\n", s1, s2);

    return 0;
}
```

Output:
aardvark is less than xylophone

strcpy

文字配列を他にコピーします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`char *strcpy(char *dest, const char *source);`

引数 この関数の引数です。

dest	char *	デスティネーション文字列
source	const char *	コピーされる文字列

注意点 `strcpy()` 関数は、`source` によって指された文字配列を、`dest` で指された文字配列にコピーします。`dest` に生成された文字配列は、同様にヌルで終了します。

`dest` と `source` によって指された配列が重複した場合、`strcpy()` の動作は定義されていません。

戻り値 `strcpy()` は `dest` の値を返します。

参照 [「memcpy」\(p279 \)](#)、[「memmove」\(p279 \)](#)、[「strncpy」\(p292 \)](#)

例 27.10 strcpy() の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    char d[30] = "";
    static char s[] = "Metrowerks";

    printf(" s=%s\n d=%s\n", s, d);
    strcpy(d, s);
    printf(" s=%s\n d=%s\n", s, d);

    return 0;
}
```

```
Output:
s=Metrowerks
d=
s=Metrowerks
d=Metrowerks
```

strcspn

他の文字配列にない文字を数えます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <string.h>
size_t strcspn(const char *s1, const char *s2);
```

引数 この関数の引数です。

s1	const char *	数える文字列
s2	const char *	検索する文字のリスト文字列

注意点 strcspn() 関数は、s2 の文字配列中に含まれない s1 の文字配列の最初からの長さを求めます。関数は、s1 の最初から文字を数えることを始めて、s2 の 1 文字が s1 の 1 文字にマッチするまで続けられます。

s1 と s2 両方ともヌルで終わる文字配列を指さなければなりません。

戻り値 `strcspn()` は、`s2` のいかなる文字にもマッチしない場合、`s1` の文字の長さを返します。

参照 [「strpbrk」\(p294\)](#)、[「strspn」\(p296\)](#)

例 27.11 `strcspn()` の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    static char s1[] = "chocolate *cinnamon* 2 ginger";
    static char s2[] = "1234*";

    printf(" s1 = %s\n s2 = %s\n", s1, s2);
    printf(" %d\n", strcspn(s1, s2));

    return 0;
}
```

Output:

```
s1 = chocolate *cinnamon* 2 ginger
s2 = 1234*
10
```

strdup

文字列を複写します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <string.h>
char *_strdup(const char *str);
char * strdup(const char *str);
```

引数 この関数の引数を以下に示します。

str	const char *	コピーされる文字列
-----	--------------	-----------

注意： この関数は `extras.c` で定義されていますが、Windows システム以外の標準ライブラリやヘッダには含まれていません。

戻り値 格納した場所へのポインタ、失敗した場合は `NULL` を返します。

注意点 Windows のルーチンでは、名前の前に下線を付けます。

参照 [「memcpy」\(p279\)](#)

strerror

文字配列にエラーメッセージを返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <string.h>
char *strerror(int errnum);
```

引数 この関数の引数です。

errnum int errno のインデックスを提供する

注意点 `strerror()` 関数は、エラーメッセージを含んだヌルで終わる文字配列へのポインタを返します。引数 `errnum` は、`strerror()` によって返されます。

戻り値 `strerror()` は、メッセージが含まれたヌルで終わる文字配列へのポインタを返します。

例 27.12 `strerror()` の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    puts(strerror(8));

    return 0;
}
```

Output:
unknown error (8)

_stricmp

文字列の大文字、小文字の違いを考慮しない比較関数です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`int _stricmp(`
`const char *s1,`
`const char *s2);`

引数 この関数の引数を以下に示します。

s1	const char *	比較される文字列
s2	const char *	比較文字列

戻り値 `stricmp` は、`str1` が `str2` より大きい場合、ゼロより大きな値を返し、`sstr2` が `tr1` より大きい場合、ゼロより小さい値を返します。等しい場合はゼロを返します。

参照 [「strcmp」\(p283\)](#)、[「_strnicmp」\(p293\)](#)

strlen

文字配列の長さを求めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`size_t strlen(const char *s);`

引数 この関数の引数です。

s1	const char *	評価する文字列
----	--------------	---------

注意点 `strlen()` 関数は、`s` によって指されたヌルで終わる文字配列の文字数を計算します。ヌル文字 (`'\0'`) は、文字数には加えられません。

戻り値 `strlen()` は、終わりを表すヌル文字を含まない文字配列中の文字数を返します。

例 27.13 strlen() の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    static char s[] = "antidisestablishmentarianism";
```

```
    printf("The length of %s is %ld.\n", s, strlen(s));

    return 0;
}
```

Output:
The length of antidisestablishmentarianism is 28.

strncasecmp

指定された長さを用いた、大文字、小文字の違いを考慮しない文字列の比較関数です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <string.h>
int strncasecmp(
    const char *s1,
    const char *s2,
    unsigned n);
```

引数 この関数の引数を以下に示します。

str1	const char *	比較される文字列
str2	const char *	比較文字列
n	unsigned int	比較の長さ

戻り値 Strncasecmp は、str1 が str2 より大きい場合、ゼロより大きな値を返し、str2 が str1 より大きい場合、ゼロより小さい値を返します。等しい場合、ゼロを返します。

参照 「[strcasecmp](#)」(p281)

strncat

文字配列に指定された文字の数だけ追加します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <string.h>
char *strncat(char *dest,
    const char *source, size_t n);
```

引数 この関数の引数です。

dest	char *	デスティネーション文字列
source	const char *	追加するソース
n	size_t	追加の最大長

注意点 `strncat()` 関数は、`source` で指された文字配列の最大 `n` 文字分を `dest` で指された文字配列へ追加します。引数 `dest` は、ヌルで終わる文字配列を指さなければなりません。引数 `source` は、ヌルで終わる文字配列を指す必要はありません。

`n` 文字が追加される前に `source` 内でヌル文字に到達すると、`strncat()` は停止します。

実行が完了した時、`strncat()` はヌル文字 (`'\0'`) を `dest` の終わりとします。

戻り値 `strncat()` は `dest` の値を返します。

参照 [「strcat」\(p281 \)](#)

例 27.14 `strncat()` の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    static char s1[100] = "abcdefghijklmnopqrstuv";
    static char s2[] = "wxyz0123456789";

    strncat(s1, s2, 4);
    puts(s1);

    return 0;
}
```

Output:
abcdefghijklmnopqrstuvwxyz

`strncmp`

指定された文字数で比較します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`int strncmp(const char *s1,`

```
const char *s2, size_t n);
```

引数 この関数の引数です。

s1	const char *	比較する文字列
s2	const char *	比較文字列
n	size_t	比較の最大長

注意点 `strncmp()` 関数は、`s1` で指された文字配列の `n` 文字分と `s2` で指された文字列の `n` 文字分を比較します。`s1` と `s2` の両方ともヌル文字で終わる文字配列でなくてもよいです。

`n` 文字分の比較が終わる前にヌル文字に到達した場合、関数は終了を待たずに比較を停止します。

戻り値 `strncmp()` は、`s1` と `s2` の最初の `n` 文字が等しい場合、ゼロの値を返し、`s1` が `s2` より小さい場合、負の値、`s1` が `s2` より大きい場合、正の値をそれぞれ返します。

参照 [「memcmp」\(p278 \)](#)、[「strcmp」\(p283 \)](#)

例 27.15 `strncmp()` の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    static char s1[] = "12345anchor", s2[] = "12345zebra";

    if (strncmp(s1, s2, 5) == 0)
        printf("%s is equal to %s\n", s1, s2);
    else
        printf("%s is not equal to %s\n", s1, s2);

    return 0;
}
```

Output:
12345anchor is equal to 12345zebra

`strncpy`

指定された文字数分をコピーします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`char *strncpy(char *dest,`
`const char *source, size_t n);`

引数 この関数の引数です。

<code>dest</code>	<code>char *</code>	デスティネーション文字列
<code>source</code>	<code>const char *</code>	コピーするソース
<code>n</code>	<code>size_t</code>	コピーの最大長

注意点 `strncpy()` 関数は、`source` で指された文字配列の最大 `n` 文字分を `dest` で指された文字配列へコピーします。`dest`、または `source` はヌルで終わる文字配列を指す必要はありません。`dest` と `source` は、重複不可です。

`n` 文字がコピーされる前に `source` 内でヌル文字 (`'\0'`) に到達すると、`strncpy()` は `dest` に `n` 文字分追加されるまでヌル文字で埋め続けます。

ヌル文字に到達する前に `source` から `n` 文字分コピーされた場合、関数はヌル文字で `dest` を終了しません。

戻り値 `strncpy()` は、`dest` の値を返します。

参照 [「memcpy」\(p279 \)](#)、[「memmove」\(p279 \)](#)、[「strcpy」\(p285 \)](#)

例 27.16 `strncpy` の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    char d[50];
    static char s[] = "123456789ABCDEFGH";

    strncpy(d, s, 9);
    puts(d);

    return 0;
}
```

Output:
123456789

`_strnicmp`

大文字、小文字の違いを考慮しない、比較の長さ指定を用いた比較関数です。

互換性 大文字、小文字の違いを考慮しない、比較の長さ指定を用いた比較関数です。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`int _strnicmp(`
 `const char *s1,`
 `const char *s2,`
 `int n);`

引数 この関数の引数を以下に示します。

s1	const char *	比較される文字列
s2	const char *	比較文字列
n	int	比較する長さの最大

戻り値 Strnicmp は、str2 より str1 が大きい場合、ゼロ以上の値を返し、str1 より str2 が大きい場合、ゼロ以下の値を返します。同じである場合、ゼロを返します。

参照 [「strcmp」\(p283\)](#)、[「_stricmp」\(p288\)](#)

strpbrk

他の文字配列での文字列の最初の出現を探します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`char *strpbrk(const char *s1, const char *s2);`

引数 この関数の引数です。

s1	const char *	検索されている文字列
s2	const char *	検索する文字のリスト

注意点 strpbrk() 関数は、s2 の文字列の 1 文字が s1 の文字列で最初に現れるところを検索します。

s1 と s2 の両方ともヌルで終わる文字配列を指さなければなりません。

戻り値 strpbrk() は、s2 の任意の文字にマッチする s1 の最初の文字へのポインタを返し、マッチしなければヌルポインタ (NULL) を返します。

参照 [「strcspn」\(p286\)](#)

例 27.17 strpbrk の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    static char s1[] = "orange banana pineapple *plum";

    static char s2[] = "%*#$";
    puts(strpbrk(s1, s2));

    return 0;
}
```

Output:
*plum

strrchr

文字が最後に見つかった場所を検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`char *strrchr(const char *s, int c);`

引数 この関数の引数です。

s	const char *	検索する文字列
c	int	検索する文字

注意点 `strrchr()` 関数は、s で指された文字配列中の c が最後に見つかった場所を探します。引数 s は、ヌルで終わる文字配列へのポインタでなければなりません。

戻り値 `strrchr()` は、見つかった文字へのポインタを返します。失敗した場合、ヌルポインタ (NULL) を返します。

参照 [「memchr」\(p276 \)](#)、[「strchr」\(p282 \)](#)

例 27.18 strrchr() の使用例

```
#include <string.h>
#include <stdio.h>
```

```
int main(void)
{
    static char s[] = "Marvin Melany Metrowerks";
    puts(strrchr(s, 'M'));

    return 0;
}
```

Output:
Metrowerks

_strrev

Strrev は文字列を反転させる関数です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`char * _strrev(char *str);`

引数 この関数の引数を以下に示します。

str	char	反転する文字列
-----	------	---------

戻り値 反転した文字列へのポインタを返します。

参照 [「strcpy」\(p285\)](#)

strspn

ある文字が他の文字列にいくつあるか数えます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`size_t strspn(const char *s1, const char *s2);`

引数 この関数の引数です。

s1	const char *	数える文字列
s2	const char *	検索する文字のリスト

注意点 `strspn()` 関数は、`s2` の文字を含む `s1` の文字列での最初からの文字数を数えます。関数は、`s1` の始めから文字を数え、`s2` にない文字が見つかるまで数え続けます。

`s1` と `s2` 両方ともヌルで終わる文字列を指さなければなりません。

戻り値 `strspn()` は、`s2` の文字にマッチする `s1` の文字数を返します。

参照 [「strpbrk」\(p294\)](#)、[「strcspn」\(p286\)](#)

例 27.19 `strspn()` の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    static char s1[] = "create *build* construct";
    static char s2[] = "create *";

    printf(" s1 = %s\n s2 = %s\n", s1, s2);
    printf(" %d\n", strspn(s1, s2));

    return 0;
}
```

Output:
s1 = create *build* construct
s2 = create *
8

strstr

文字列を他の文字列内で検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`char *strstr(const char *s1, const char *s2);`

引数 この関数の引数です。

<code>s1</code>	<code>const char *</code>	検索する文字列
<code>s2</code>	<code>const char *</code>	検索する文字列

注意点 `strstr()` 関数は、`s2` で指される文字配列が、`s1` で指された文字配列で最初に見かるところを検索します

`s1` と `s2` 共にヌル (`\0`) で終わる文字配列を指さなければなりません。

戻り値 `strstr()` は、`s1` で `s2` が最初に見つかったところへのポインタを返し、見つからない場合、ヌルポインタ (`NULL`) を返します。

参照 [「memchr」\(p276 \)](#)、[「strchr」\(p282 \)](#)

例 27.20 `strstr()` の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    static char s1[] = "tomato carrot onion";
    static char s2[] = "on";
    puts(strstr(s1, s2));

    return 0;
}
```

Output:
onion

`strtok`

文字配列内のトークンを抽出します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`char *strtok(char *str, const char *sep);`

引数 この関数の引数です。

<code>str</code>	<code>char *</code>	分割する文字列
<code>sep</code>	<code>const char *</code>	文字列のセパレータ

注意点 `strtok()` 関数は、`str` によって指された文字列をトークン化します。引数 `sep` は、トークン分離文字を含む文字配列を指します。`str` のトークンは、`strtok()` を連続的に呼び出すことで抽出されます。

`strtok()` の最初の呼び出しは、`sep` に存在しない最初の文字を `str` で探すことになります。関数は、最初のトークンの始まりを指すポインタを返します。文字が見つからない場合、`strtok()` は、ヌルポインタ (NULL) を返します。

最初の呼び出しで `strtok()` がトークンを見つけた場合、次の呼び出しでは次のトークンを探します。

関数は、`str` のトークンの文字をスキップして `sep` の文字が見つかるまで検索します。この文字は、トークンの文字列を終わらせるためヌル文字で上書きされます。それによって、文字配列の内容は変更されます。この関数は、ヌル文字の後の次のトークンに対する文字を指す関数自身のポインタを保持します。

引き続いてのトークンの検索は、内部ポインタから同じ方法で続けられます。NULL の引数 `str` で `strtok()` を引き続き呼び出すと、元の `str` の文字配列の残りのトークンへのポインタを返します。トークンが存在しない場合、`strtok()` はヌルポインタを返します。引数 `sep` は、`strtok()` のそれぞれの呼び出しで違えることができます。

`str` と `sep` の両方ともヌル文字で終わる文字配列でなければなりません。

戻り値 最初に呼び出された `strtok()` は、`str` の最初のトークンへのポインタを返します。トークンが見つからない場合、ヌルポインタを返します。

NULL の引数 `str` を使って引き続き `strtok()` を呼び出すと、次のトークンへのポインタを返します。トークンが存在しない場合、ヌルポインタ (NULL) を返します。

`strtok()` は、`str` によって指された文字配列を変更します。

例 27.21 `strtok()` の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    static char s[50] = "(ape+bear)*(cat+dog)";
    char *nexttok;

    // strtok() の最初の呼び出し
    puts(strtok(s, "()+*"));

    nexttok = strtok(NULL, "()+*");
    puts(nexttok);

    nexttok = strtok(NULL, "()+*");
    puts(nexttok);

    nexttok = strtok(NULL, "()+*");
    puts(nexttok);

    return 0;
```

```
}

```

Output:
ape
bear
cat
dog

strxfrm

地域指定文字配列の変換。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <string.h>`
`size_t strxfrm(char *dest,`
`const char *source, size_t n);`

引数 この関数の引数です。

dest	char *	デスティネーション文字列
source	const char *	変換するソース
n	size_t	変換する最大長

注意点 `strxfrm()` 関数は、`source` に指された文字配列を `dest` に指された文字配列へコピーします。それぞれの文字の変換は、`locale.h` で定義された地域文字セット (locale character set) に準じます。

`strxfrm()` の CodeWarrior C での実装では、`strncpy()` 関数を使って `source` で指された文字配列を `dest` で指された文字配列へ最大 `n` 文字分コピーします。これは、文字列ライブラリに含まれ、ANSI C の標準ライブラリ仕様に準拠しています。

戻り値 `strxfrm()` は、`source` から受け取った `dest` の長さを返します。

参照 [「地域設定情報の仕様」\(p73\)](#)、[「strcpy」\(p285\)](#)

例 27.22 strxfrm() の使用例

```
#include <string.h>
#include <stdio.h>

int main(void)
{
    char d[50];
    static char s[] = "123456789ABCDEFGH";
```

```

size_t result;

result = strxfrm(d, s, 30);

printf("%d characters copied: %s\n", result, d);

return 0;
}

```

Output:
16 characters copied: 123456789ABCDEFG

_strupr

Strupr は、文字列を大文字に変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `char *_strupr(char *str);`

引数 この関数の引数を以下に示します。

str	char	変換される文字列
-----	------	----------

戻り値 変換された文字列へのポインタを返します。

参照 [「toupper」\(p46 \)](#)、[「tolower」\(p45 \)](#)

第 28 章 time.h

time.h ヘッドファイルは、コンピュータのシステムクロックにアクセスして、日付、時間変換関数ならびに時間の書式関数を提供します。

time.h の概要

以下の内容について説明します。

- [「struct tm」\(p304\)](#) : 時間データを記憶する構造体
- [「tzname」\(p305\)](#) : 時間帯の省略形が記憶される配列
- [「asctime」\(p305\)](#) : tm 構造体の型から文字配列へ変換する
- [「clock」\(p306\)](#) : コンピュータを初めて起動してからの時間を求める
- [「ctime」\(p307\)](#) : time_t 型から文字配列へ変換する
- [「difftime」\(p308\)](#) : 2 つの時間の違いを調べる
- [「gmtime」\(p309\)](#) : グリニッジ標準時を求める
- [「localtime」\(p310\)](#) : 現地時間を求める
- [「mktime」\(p310\)](#) : tm 構造体から time_t 型へ変換する
- [「strftime」\(p311\)](#) : 文字列バッファに日付を記憶する
- [「strdate」\(p311\)](#) : データを文字列バッファへ保存する
- [「time」\(p316\)](#) : 設定された時間から秒数を求める
- [「tzset」\(p317\)](#) : アプリケーションの時間帯を修正する

日付と時間

time.h ヘッドファイルは、コンピュータのシステムクロックにアクセスして、日付、時間変換関数ならびに時間の書式関数を提供します。

3 つのデータの型 (clock_t、time_t、tm) が time.h で定義されます。

clock_t 型は数値で、システムに依存した型を持ち clock() 関数によって返されます。

time_t 型は、システムに依存した型で、カレンダーの日付と時間を表現するため利用されます。

注意： ANSI/ISO C 規格は開始日付を指定していません。そのため、MSL C ライブラリでは 1990 年 1 月 1 日を選択しました。これらの関数は特定の API 時間関数と一緒に使用しないでください。OS 特有のヘッダ (time.mac.h など) では定数の変換が可能です。

struct tm

struct tm 型は、カレンダーの日付と時間のそれぞれの部分フィールドを持ちます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <time.h>`
`struct tm {`
 `int tm_sec;`
 `int tm_min;`
 `int tm_hour;`
 `int tm_mday;`
 `int tm_mon;`
 `int tm_year;`
 `int tm_wday;`
 `int tm_yday;`
 `int tm_isdst;`
`};`

注意点 tm 構造体のメンバを「[Tm 構造体のメンバ](#)」(p304) に列挙します。

注意： tm_isdst フラグは、夏時間である場合は正の値であり、そうでなければゼロ、そのような情報が利用できなければ負の値となります。

表 28.1 Tm 構造体のメンバ

フィールド	説明	範囲：最小 - 最大
int tm_sec	毎分中の秒数	0 - 59
int tm_min	毎時間中の分数	0 - 59
int tm_hour	午前 0 時からの時間数	0 - 23
int tm_mday	各月の中での日数	1 - 31
int tm_mon	1 月からの月数	0 - 11
int tm_year	1900 年からの年数	
int tm_wday	日曜日からの日数	0 - 6

int tm_yday	1 月からの日数	0 - 365
int tm_isdst	夏時間用のフラグ	

tzname

tzname array は、現地の標準時間と DST の名前（省略形）を保持します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <time.h>
extern char *tzname[2];
```

参照 [「tzset」\(p317\)](#)

asctime

tm 構造体から文字配列へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <time.h>
char *asctime(const struct tm *timeptr);
```

引数 この関数の引数です。

timeptr const struct tm * 時間情報を持つ tm 構造体へのポインタ

注意点 asctime() 関数は、timeptr によって指された tm 構造体から文字配列へ変換します。asctime() と ctime() 関数は、同じカレンダーの時間の書式を使います。この書式は strftime() の書式として表現され、%a %b %d %H:%M: %S %Y となります。

戻り値 asctime() は、tm 構造体を変換したヌルで終わる文字列のポインタを返します。

参照 [「ctime」\(p307\)](#)、[「strftime」\(p311\)](#)

例 28.1 asctime() の使用例

```
#include <time.h>
#include <stdio.h>

int main(void)
```

```
{
    time_t systime;
    struct tm *currrtime;

    systime = time(NULL);
    currrtime = localtime(&systime);

    puts(asctime(currrtime));

    return 0;
}
```

Output:
Tue Nov 30 12:56:05 1993

clock

システムが動作し続けている総時間を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <time.h>`
`clock_t clock(void);`

変数 なし。

注意点 `clock()` 関数は、コンピュータシステムが動作を開始してからの総時間を返します。秒数で時間を計算するために、`clock_t` の値は `time.h` で定義されたマクロ、`CLOCKS_PER_SEC` で割られます。

戻り値 `clock()` は、システムが動作してからの時間を表現する `clock_t` 型の値を返します。

例 28.2 clock() の使用例

```
#include <time.h>
#include <stdio.h>

int main(void)
{
    clock_t uptime;

    uptime = clock() / CLOCKS_PER_SEC;

    printf("I was booted %ul seconds ago.\n", uptime);
}
```

```
    return 0;
}
```

Output:
I was booted 24541 seconds ago.

ctime

time_t 型を文字配列に変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <time.h>`
`char *ctime(const time_t *timer);`

引数 この関数の引数です。

timer const time_t * time_t 変数のアドレス

注意点 ctime() 関数は、asctime() で使用される同じ書式を用いて time_t 型を文字配列に変換します。

戻り値 ctime() は、変換された time_t 型を保持するヌルで終わる文字配列のポインタを返します。

参照 [「asctime」\(p305 \)](#)、[「strftime」\(p311 \)](#)

例 28.3 ctime() の使用例

```
#include <time.h>
#include <stdio.h>

int main(void)
{
    time_t systime;

    systime = time(NULL);
    puts(ctime(&systime));

    return 0;
}
```

Output:
Wed Jul 20 13:32:17 1994

difftime

2 つの time_t 型の差を計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <time.h>`
`double difftime(time_t t1, time_t t2);`

引数 この関数の引数です。

t1	time_t	比較する time_t 変数
t2	time_t	比較する time_t 変数

戻り値 `difftime()` は、秒で表現された t1 から t2 を引いた差を返します。

例 28.4 difftime の使用例

```
#include <time.h>
#include <stdio.h>

int main(void)
{
    time_t t1, t2;
    struct tm *currtime;
    double midnight;

    time(&t1);
    currtime = localtime(&t1);

    currtime->tm_sec = 0;
    currtime->tm_min = 0;
    currtime->tm_hour = 0;
    currtime->tm_mday++;

    t2 = mktime(currtime);

    midnight = difftime(t1, t2);
    printf("There are %f seconds until midnight.\n",midnight);

    return 0;
}
```

```
}
```

Output:
There are 27892.000000 seconds until midnight.

gmtime

time_t の値をグリニッジ標準時の新しい名前、万国標準時 (UTC) に変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
-------------	-------------	-----------------	---------------	----------------	--------------	--

プロトタイプ #include <time.h>
struct tm *gmtime(const time_t *timer);

引数 この関数の引数です。

timer const time_t * time_t 変数のアドレス

注意点 gmtime 関数は、timer によって指されたカレンダーの時間を、UTC で表現された区分された時間に変換します。

戻り値 gmtime() 関数は、区分された時間を指すポインタを返します。

例 28.5 gmtime の使用例

```
#include <time.h>
#include <stdio.h>

int main(void)
{
    time_t systime;
    struct tm *utc;

    systime = time(NULL);
    utc = gmtime(&systime);

    printf("Universal Coordinated Time:\n");
    puts(asctime(utc));

    return 0;
}
```

Output:
Universal Coordinated Time:
Thu Feb 24 18:06:10 1994

localtime

time_t 型から struct tm 型へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <time.h>`
`struct tm *localtime(const time_t *timer);`

引数 この関数の引数です。

timer const time_t * time_t 変数のアドレス

注意点 localtime() 関数は、timer で指された time_t 型を変換して、内部の struct tm 型へのポインタとして返します。struct tm ポインタは static 変数であり、localtime() が呼ばれる度に上書きされます。

戻り値 localtime() は timer を変換して struct tm へのポインタを返します。

参照 [「mktime」\(p310\)](#)

使用に関して [「difftimeの使用例」\(p308\)](#) の例を参照してください。

mktime

struct tm の項目を time_t 型に変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <time.h>`
`time_t mktime(struct tm *timeptr);`

引数 この関数の引数です。

timeptr struct tm * tm 構造体のアドレス

注意点 mktime() 関数は、struct tm 型を変換し time_t 型として返します。

この関数は必要に応じて timeptr のフィールドを調整します。tm_sec、tm_min、tm_hour、tm_day は、それらが最大値より大きければ適当な桁上げが計算されるよう処理されます。例えば timeptr->tm_min が 65 である場合、timeptr->tm_hour は 1 増やされ、timeptr->min は 5 に設定されます。

関数は、timeptr->tm_wday と timeptr->tm_yday に関しても正しい値を計算します。

戻り値 mktime() は、tm 構造体を time_t 型として変換します。

参照 [「localtime」\(p310\)](#)

例 28.6 使用例

[「difftime の使用例」\(p308\)](#) の例を参照してください。

`_strdate`

strdate 関数は、提供されたバッファに日付を記憶します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <time.h>`
`char *_strdate(char *str);`

引数 この関数の引数です。

str char * 日付を記憶する char 型の文字列

戻り値 関数は引数 str へのポインタを返します。

注意点 この関数は、バッファ内に日付を mm/dd/yy の書式で記憶させます。ここでのバッファは、少なくとも 9 文字でなければなりません。

参照 [「strftime」\(p311\)](#)

`strftime`

tm 構造体の書式です。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <time.h>`
`size_t strftime(char *s, size_t maxsize,`
`const char *format,`
`const struct tm *timeptr);`

引数 この関数の引数です。

s	char *	書式化する文字列
format	const char *	書式化文字列
timeptr	const struct tm *	time 構造体のアドレス

注意点 `strftime()` 関数は、`tm` 構造体をプログラマが指定した書式の文字配列に変換します。

引数 `s` は書式化された時間を持つ配列へのポインタです。

引数 `maxsize` は書式化された文字配列の最大の長さを指定します。

引数 `timeptr` は、変換や書式化するためのカレンダーの時間を持つ `tm` 構造体を指します。

引数 `format` は、`printf()` 関数の書式文字列と同様のテキストと書式指定を持つ文字配列を指します。書式指定子は、パーセント符号が前に付けられます。パーセント符号を 2 つ続けることで (%%) 1 つの % を出力します。

指定した値が通常の範囲外であれば、その文字は格納されて無指定になります。

注意： 書式指定子のリストに関しては「[strftime\(\) の変換文字](#)」(p312) をご覧ください。

表 28.2 `strftime()` の変換文字

Char	説明
a	曜日の名前の省略形
A	曜日の完全な名前
b	月の名前の省略形
B	月の完全な名前
c	<code>strftime()</code> は %x%X の書式文字列と同様な書式化を行う
C	100 で割った年を、10 進数値として整数に省略する
d	各月の 10 進数の日
D	月、日、年：%m/%d/%Y と等価
H	00 から 23 までの 10 進数の時間 (24 時間制)

Char	説明
e	10 進数の月と日：1 桁の場合先頭にスペースが付加される
F	年、月、日：%Y-%m-%d と等価
g	10 進数の週単位の年の下 2 桁（例：03 99）
G	10 進数の週単位の年
h	%b と等価な月の名前
H	01 から 23 までの 10 進数の時間（24 時間制）
I	01 から 12 までの 10 進数の時間（12 時間制）
j	001 から 366 までの 10 進数の各年の日
m	01 から 12 までの 10 進数の月
M	00 から 59 までの 10 進数の分
n	改行キャラクタ
p	AM または PM
r	12 時間制のクロック
R	%H:%M と等価な時間と分
S	00 から 59 までの 10 進数の秒
t	水平タブキャラクタ
T	%H:%M:%S と等価な時間、分、秒
u	0 から 7 までの 10 進数での週日の数（月曜日は 1）
U	00 から 53 までの 10 進数での 1 年の週の数（日曜日を週の始めとする）
w	0 から 6 までの 10 進数での週日の数（日曜日は 0）
W	00 から 51 までの 10 進数での 1 年の週の数（月曜日を週の始めとする）
x	現在の地域の日付表現
X	現在の地域での時間表現
y	年の最後の 2 桁の 10 進数字
Y	10 進数での世紀
z	時間帯の名前（分からない場合何も表示しない）
Z	地域時間帯の名前または省略形、時間帯が未知の場合はキャラクタなし
%	パーセント符号（を表示）

戻り値 `strftime()` 関数は、文字列引数 `s` 中のヌル文字を含んだ総文字数が、引数 `maxlen` の値よりも小さい場合、文字列引数 `s` の総文字数を返します。大きい場合、0 を返します。

例 28.7 strftime() の使用例

```
#include <time.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    time_t lclTime;
    struct tm *now;
    char ts[256]; /* time string */

    lclTime = time(NULL);
    now = localtime(&lclTime);

    strftime(ts, 256,
        "Today's abr.name is %a", now);
    puts(ts);

    strftime(ts, 256,
        "Today's full name is %A", now);
    puts(ts);

    strftime(ts, 256,
        "Today's aabr.month name is %b", now);
    puts(ts);

    strftime(ts, 256,
        "Today's full month name is %B", now);
    puts(ts);

    strftime(ts, 256,
        "Today's date and time is %c",now);
    puts(ts);
    strftime(ts, 256,
        "The day of the month is %d", now);
    puts(ts);

    strftime(ts, 256,
        "The 24-hour clock hour is %H",now);
    puts(ts);

    strftime(ts, 256,
        "The 12-hour clock hour is %H", now);
    puts(ts);

    strftime(ts, 256,
        "Today's day number is %j", now);
    puts(ts);
```

```
    strftime(ts, 256,  
"Today's month number is %m", now);  
    puts(ts);  
  
    strftime(ts, 256,  
"The minute is %M", now);  
    puts(ts);  
  
    strftime(ts, 256,  
"The AM/PM is %p", now);  
    puts(ts);  
  
    strftime(ts, 256,  
"The second is %S", now);  
    puts(ts);  
  
    strftime(ts, 256,  
"The week number of the year,\nstarting on a Sunday is %U", now);  
    puts(ts);  
  
    strftime(ts, 256,  
"The number of the week is %w", now);  
    puts(ts);  
  
    strftime(ts, 256, "The week number of the year,\nstarting on a Monday is %W", now);  
    puts(ts);  
  
    strftime(ts, 256, "The date is %x", now);  
    puts(ts);  
  
    strftime(ts, 256, "The time is %X", now);  
    puts(ts);  
  
    strftime(ts, 256,  
    "The last two digits of the year are %y", now);  
    puts(ts);  
  
    strftime(ts, 256, "The year is %Y", now);  
    puts(ts);  
  
    strftime(ts, 256, "%Z", now);  
    if (strlen(ts) == 0)  
        printf("The time zone cannot be determined\n");  
    else  
        printf("The time zone is %s\n", ts);  
  
    return 0;  
}
```

```
Results
Today's abr.name is Mon
Today's full name is Monday
Today's aabr.month name is Sep
Today's full month name is September
Today's date and time is Monday, 27 September, 1999  04:24:20 PM
The day of the month is 27
The 24-hour clock hour is 16
The 12-hour clock hour is 16
Today's day number is 270
Today's month number is 09
The minute is 24
The AM/PM is PM
The second is 20
The week number of the year,starting on a Sunday is 39
The number of the week is 1
The week number of the year,starting on a Monday is 39
The date is 27 September, 1999
The time is 16:24:20
The last two digits of the year are 99
The year is 1999
The time zone cannot be determined
```

time

現在のシステムカレンダーの時間を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

```
プロトタイプ #include <time.h>
time_t time(time_t *timer);
```

引数 この関数の引数です。

timer time_t * time_t 変数のアドレス

注意点 time() 関数はシステムカレンダーの時間を返します。timer がヌルポインタでない場合、
カレンダーの時間がその指し示すところに割り当てられます。

戻り値 time() は、システムの現在のカレンダーの時間を返します。

例 28.8 time() の使用例

```
#include <time.h>
#include <stdio.h>

int main(void)
{
    time_t systime;
    systime = time(NULL);

    puts(ctime(&systime));

    return 0;
}
```

Output:
Tue Nov 30 13:06:47 1993

tzset

関数 `tzset()` は TZ 環境変数の値を読み込み、プログラムの時間帯機能を修正します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <time.h>`
`void tzset(void);`

引数 なし。

注意点 関数 `tzset()` は TZ 環境変数の値を読み込み、プログラムの時間帯機能を修正します。

参照 [「tzname」\(p305\)](#)

第 29 章 unistd.h

unistd.h ヘッダファイルは、UNIX からプログラムを移植するためのいくつかの使い易い関数を含んでいます。

unistd.h の概要

unistd.h ヘッダファイルの関数は多くの UNIX ライブラリの関数と同様です。しかし、UNIX と Mac OS のオペレーティングシステムは、基本的に異なる部分があり同じとはなりません。関数の説明では、何が違うのかという部分を説明します。

以下に unistd.h の関数の機能を説明します。

[「chdir」\(p320\)](#): ディレクトリを変更する

[「close」\(p322\)](#): open によってオープンされたファイルをクローズする

[「cuserid」\(p324\)](#): 現在のユーザの ID を検索する

[「exec」\(p325\)](#): プログラムから他のプログラムを実行する

[「getcwd」\(p327\)](#): 現在の作業ディレクトリを取得する

[「getlogin」\(p328\)](#): ログイン名を返す

[「getpid」\(p328\)](#): プロセス ID を返す

[「isatty」\(p330\)](#): ファイル ID が端末へ接続されているかどうかを判定する

[「lseek」\(p331\)](#): open によってオープンされた場合にシークする

[「read」\(p332\)](#): open によってオープンされた場合に読み込む

[「rmdir」\(p333\)](#): ディレクトリやフォルダを削除する

[「sleep」\(p333\)](#): プログラムを休止する

[「ttyname」\(p334\)](#): 端末 ID を判定する

[「unlink」\(p335\)](#): ファイルを削除する

unistd.h と UNIX の互換性

一般的に、新しいプログラムでは、これらの関数を使わず、代わりに API 本来の対応するものを利用するとよいでしょう。

注意： UNIX や DOS のプログラムを移植するのである場合、UNIX と互換のある他のヘッダの関数が必要になるかもしれません。

chdir

現在のディレクトリを変更します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unistd.h>`
`int chdir(const char *path);`

引数 この関数の引数です。

path char * 新しいパス名

注意点 関数 `chdir()` は、あるディレクトリから異なるディレクトリやフォルダへ変更する場合に利用されます。使用例は、[「chdir\(\) の使用例」\(p320\)](#) で説明します。

戻り値 `chdir()` は、成功した場合ゼロを返します。失敗した場合、負の値を返し、`errno` を設定します。

参照 [「errno.h の概要」\(p49\)](#)

例 29.1 `chdir()` の使用例

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stat.h>

#define SIZE FILENAME_MAX
#define READ_OR_WRITE0x0 /* UNIX モードの模倣 */

int main(void)
{
    char folder[SIZE];
    char curFolder[SIZE];
    char newFolder[SIZE];
    int folderExisted = 0;

    /* 現在のフォルダあるいはディレクトリの名前を取得 */
    getcwd( folder, SIZE );
    printf("The current Folder is: %s", folder);
```



```
/* 新しいサブフォルダを生成 */
/* Mac ではモードの引数が無視されることに注意 */
sprintf(newFolder, "%s%s", folder, "Sub" );
if( mkdir(newFolder, READ_OR_WRITE ) == -1 )
{
    printf("\nFailed to Create folder");
    folderExisted = 1;
}

/* 新しいフォルダへ変更 */
if( chdir( newFolder) )
{
    puts("\nCannot change to new folder");
    exit(EXIT_FAILURE);
}

/* 新しいフォルダを見せる */
getcwd( curFolder, SIZE );
printf("\nThe current folder is: %s", curFolder);

/* 前のフォルダへ戻る */
if( chdir(folder) )
{
    puts("\nCannot change to old folder");
    exit(EXIT_FAILURE);
}

/* 新しいフォルダを見せる */
getcwd( curFolder, SIZE );
printf("\nThe current folder is again: %s", curFolder);

if (!folderExisted)
{
    /* 新しく生成されたディレクトリを削除 */
    if (rmdir(newFolder))
    {
        puts("\nCannot remove new folder");
        exit(EXIT_FAILURE);
    }
    else
        puts("\nNew folder removed");
}

/* 存在しないディレクトリの移動を試みる */
if (chdir(newFolder))
    puts("Cannot move to non-existant folder");
}
else puts("\nPre-existing folder not removed");
```

```
    return 0;
}
```

Output

```
The current Folder is: Macintosh HD:C Reference:
The current folder is: Macintosh HD:C Reference:Sub:
The current folder is again: Macintosh HD:C Reference:
New folder removed
Cannot move to non-existant folder
```

close

オープンしたファイルをクローズします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <unistd.h>
int close(int fildes);
```

引数 この関数の引数です。

fildes	int	ファイルの説明
--------	-----	---------

注意点 `close()` 関数は、引数によって指定されたファイルをクローズします。引数は `open()` によって返された値です。使用例は [「close\(\) の使用例」\(p322\)](#) に示します。

戻り値 成功した場合、`close()` はゼロを返します。失敗した場合、`close()` は、負の値を返し、`errno` を設定します。

参照 [「open」\(p56\)](#)、[「fclose」\(p173\)](#)、[「errno」\(p49\)](#)

例 29.2 close() の使用例

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

#define SIZE FILENAME_MAX
#define MAX 1024

char fname[SIZE] = "DonQ.txt";
```

```
int main(void)
{
    int fdes;
    char temp[MAX];
    char *Don = "In a certain corner of la Mancha, the name of\n\
which I do not choose to remember,...";
    char *Quixote = "there lived\nnone of those country\
gentlemen, who adorn their\nhalls with rusty lance\
and worm-eaten targets.";

    /* printfのための NULL で終わる temp 配列 */
    memset(temp, '\\0', MAX);

    /* ファイルのオープン */
    if((fdes = open(fname, O_RDWR | O_CREAT ))== -1)
    {
        perror("Error ");
        printf("Can not open %s", fname);
        exit( EXIT_FAILURE);
    }

    /* ファイルへ書き込む */
    if( write(fdes, Don, strlen(Don)) == -1)
    {
        printf("%s Write Error\n", fname);
        exit( EXIT_FAILURE );
    }

    /* ... 文字に上書きするために戻る */
    if( lseek( fdes, -3L, SEEK_CUR ) == -1L)
    {
        printf("Seek Error");
        exit( EXIT_FAILURE );
    }

    /* ファイルへ書き込む */
    if( write(fdes, Quixote, strlen(Quixote)) == -1)
    {
        printf("Write Error");
        exit( EXIT_FAILURE );
    }

    /* 読み込むためにファイルの最初に移動する */
    if( lseek( fdes, 0L, SEEK_SET ) == -1L)
    {
        printf("Seek Error");
        exit( EXIT_FAILURE );
    }
}
```

```
    }

    /* ファイルの読み込み */
    if( read( fdes, temp, MAX ) == 0)
    {
        printf("Read Error");
        exit( EXIT_FAILURE);
    }

    /* ファイルのクローズ */
    if(close(fdes))
    {
        printf("File Closing Error");
        exit( EXIT_FAILURE );
    }

    puts(temp);

    return 0;
}
```

Result
In a certain corner of la Mancha, the name of
which I do not choose to remember,there lived
one of those country gentlemen, who adorn their
halls with rusty lance and worm-eaten targets.

cuserid

現在のユーザの ID を検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <unistd.h>
char *cuserid(char *string);

引数 この関数の引数です。

string char * ユーザー ID の文字列

注意点 関数 cuserid() は、現在のプロセスに関するユーザ名を返します。引数 string が NULL である場合、ファイル名は内部バッファに記憶されます。NULL ではない場合、少なくとも FILENAME_MAX の大きさでなければなりません。使用例を「[cuserid\(\) の使用例](#)」(p325) に示します。

注意： MacOS では、ログイン名が返されます。

戻り値 `cuserid()` 現在のユーザ ID への文字型のポインタを返します。

注意： MacOS では、ユーザ名は利用者 & グループコントロールパネルを利用して設定されます。

例 29.3 `cuserid()` の使用例

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    char *c_id = NULL;
    printf("The current user ID is %s\n",cuserid(c_id));

    return 0;
}
```

Result
The current user ID is Metrowerks

exec

子プロセスをロードして現在のプログラムメモリで実行します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

注意点 Mac OS では、全ての exec ファミリは `exec()` を用いて `pass` を呼び出します。引数 `passing(argc, argv)` は、Mac OS アプリケーションでは存在しないからです。

プロトタイプ `#include <unistd.h>`

```
int exec(const char *path, ...);
int execl(const char *path, ...);
int execlp(const char *path, ...);
int execlpe(const char *path, ...);
int execvp(const char *path, ...);
int execevp(const char *path, ...);
int exeevp(const char *path, ...);
```

注意： MacOS では、全ての exec 型は exec() を通じて pass を呼び出します。引数渡し (argc、argv) は、Mac OS アプリケーションでは存在しないからです。

引数 この関数の引数です。

path const char * 実行するコマンドラインのパス名
... 引数の変数リスト

表 29.1 exec() 型の関数

UNIX 関数	Mac OS システム
#define execl	exec
#define execv	exec
#define execl_e	exec
#define execve	exec
#define execl_p	exec
#define execvp	exec

指定されたアプリケーションが実行され、成功した場合、終了します。使用例を「[exec\(\) の使用例](#)」(p326) に示します。

戻り値 exec() が成功した場合、ゼロを返します。exec() が失敗した場合、負の値を返し、errno にエラーに応じて値を設定します。

注意： SIOUX を Mac OS で用いた場合、これらの設定は、自動的に SIOUX プログラムをクローズします。asktosaveonclose は、2 番目の printf 文が呼ばれず、元々 printf() 文が呼ばれることを示すためのデフォルト値が記憶されています。

参照 「[SIOUX と WinSIOUX の概要](#)」(p143)、 「[errno.h の概要](#)」(p49)

例 29.4 exec() の使用例

```
#include <stdio.h>
#include <SIOUX.h>
#include <unistd.h>

#define SIZE FILENAME_MAX
char appName[SIZE] = "Macintosh HD:SimpleText";

int main(void)
{
```

```

        SIOUXSettings.autocloseonquit = 1;
        SIOUXSettings.asktosaveonclose = 1;

        printf("Original Program\n");
        exec(appName);
        printf("program terminated"); /* 表示されない */

        return 0;
    }

```

```

result
Display "Original Program"
after the close of the program
the SimpleText application is launched

```

getcwd

現在のディレクトリを取得します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unistd.h>`
`char *getcwd(char *buf, int size);`

引数 この関数の引数です。

buf	char	現在の作業ディレクトリを含むバッファ
size	int	バッファのサイズ

注意点 関数 `getcwd()` には、2 つの引数があります。1 つは、ディレクトリのパスの名前が全部記憶できる大きさのバッファで、もう 1 つはバッファサイズです。

戻り値 成功した場合、`getcwd()` はバッファへのポインタを返します。失敗した場合、`getcwd()` は `NULL` を返し、`errno` を設定します。

参照 [「errno.h の概要」\(p49\)](#)

例 29.5 getcwd の使用例

[「chdir\(\) の使用例」\(p320\)](#) を参照してください。

getlogin

プログラムを開始させたユーザの名前を検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unistd.h>`
`char *getlogin(void);`

引数 なし。

注意点 関数 `getlogin()` は、プログラムをスタートさせたユーザ名を検索します。使用例を「[getlogin\(\) の使用例](#)」(p328) に示します。

注意： Mac にはログインは無いですが、この関数は所有者名を利用者 & グループコントロールパネルから値を返します。

戻り値 `getlogin()` は文字型のポインタを返します。

例 29.6 getlogin() の使用例

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    printf("The login name is %s\n", getlogin());

    return 0;
}
```

```
result
The login name is Metrowerks
```

getpid

プロセスの識別番号を検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unistd.h>`

表 29.2 `getpid()` マクロ

マクロ		表現
<code>#define getpid()</code>		プロセス ID
<code>#define getppid()</code>		親プロセス ID
<code>#define getuid()</code>		実際のユーザ ID
<code>#define geteuid()</code>		効果のあるユーザ ID
<code>#define getgid()</code>		実際のグループ ID
<code>#define getegid()</code>		効果のあるグループ ID
<code>#define getpgrp()</code>		プロセスグループ ID

引数 なし。

注意点 `getpid()` 関数は、呼んだプロセスの唯一の番号（プロセス ID）を返します。使用例を「[getpid\(\) の使用例](#)」（p329）に示します。

戻り値 `getpid()` は整数の値を返します。

注意： `getpid()` に関係するさまざまな型の関数は、Mac においては実際のところ何の意味も持ちません。返された値は、UNIX での典型的なユーザープロセスのために意味があるかもしれません。

戻り値 `getpid()` は常時ある値を返します。エラーはありません。

例 29.7 `getpid()` の使用例

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    printf("The process ID is %d\n", getpid());

    return 0;
}
```

Result
The process ID is 9000

isatty

指定された file_id を判定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unistd.h>`
`int isatty(int fildes);`

引数 この関数の引数です。

fildes	int	ファイルの説明
--------	-----	---------

注意点 関数 `isatty()` は、指定された file_id がコンソールに接続されているかどうか、または、実際はリダイレクションかどうかを判定します。使用例を [「isatty\(\)、ttyname\(\) の使用例」\(p330\)](#) に示します。

戻り値 `isatty()` は、ファイルがコンソールへ接続されていれば非ゼロの値を返します。実際は、入力 / 出力のリダイレクションである場合、ゼロを返します。

参照 [「ccommand」\(p25\)](#)

例 29.8 isatty()、ttyname() の使用例

```
#include <console.h>
#include <stdio.h>
#include <unistd.h>
#include <unix.h>

int main(int argc, char **argv)
{
    int i;
    int file_id;
    argc = ccommand(&argv);

    file_id = isatty(fileno(stdout));
    if(!file_id )
    {
        for (i=0; i < argc; i++)
            printf("command line argument [%d] = %s\n",
                i, argv[i]);
    }
    else printf("Output to window");

    printf("The associated terminal is %s",
        ttyname(file_id) );
}
```

```
    return 0;
}
```

```
Result if file redirection is chosen using the command line
arguments
Metrowerks CodeWarrior.
Written to file selected:

command line argument [0] = CRef
command line argument [1] = Metrowerks
command line argument [2] = CodeWarrior
The associated terminal is SIOUX
```

lseek

ファイルストリーム上のある位置へシークします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unistd.h>`
`long lseek(int fildes, long offset, int origin);`

引数 この関数の引数です。

fildes	int	ファイルの説明
offset	long	移動するオフセットのバイト数
origin	int	シークの開始地点

注意点 関数 `lseek()` は、指定された最初の位置から指定された `offset` バイト分のところにファイル位置を設定します。

`origin` は、[「lseek のオフセット位置」\(p331\)](#) 中で説明されている位置のいずれかでなければなりません。

表 29.3 lseek のオフセット位置

マクロ	意味
SEEK_SET	ファイルの始め
SEEK_CUR	現在の位置
SEEK_END	ファイルの終わり

戻り値 成功した場合、`lseek()` はオフセットのバイト数を返します。失敗した場合、`long` の整数型の負の値を返します。

参照 [「fseek」\(p201\)](#)、[「ftell」\(p204\)](#)、[「open」\(p56\)](#)

例 29.9 `lseek()` の使用例

[「close\(\) の使用例」\(p322\)](#) を参照してください。

read

書式化されていない入力/出力用にオープンされたファイルストリームから読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <unistd.h>
int read(int fildes, char *buf, int count);
```

引数 この関数の引数です。

fildes	int	ファイルの説明
buf	char *	読み込んだデータを保存するバッファ
count	int	読み込むバイトの最大サイズ

注意点 関数 `read()` は、引数 `count` で指定されたバイト数分、ファイルから文字バッファへコピーします。ファイルは、現在位置から読み込みを開始します。読み込み位置の終わりに移動したとき、動作は完了します。

注意： この関数は、`unistd.h:write()` と `fcntl.h:open()` のみ組み合わせて使うべきです。

戻り値 `read()` は、ファイルから実際に読み込んだバイト数を返します。エラーの場合、負の値が返され、`errno` が設定されます。

参照 [「fread」\(p194\)](#)、[「open」\(p56\)](#)

例 29.10 `read()` の使用例

[「close\(\) の使用例」\(p322\)](#) を参照してください。

rmdir

ディレクトリ、またはフォルダを削除します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unistd.h>`
`int rmdir(const char *path);`

引数 この関数の引数です。

path const char * 削除されたディレクトリのパス名

注意点 `rmdir()` 関数は、引数によって指定されたディレクトリ（フォルダ）を削除します。

戻り値 成功した場合、`rmdir()` はゼロを返し、失敗した場合、`rmdir()` は負の値を返し、`errno` を設定します。

参照 [「mkdir」\(p160\)](#)、[「errno」\(p49\)](#)

例 29.11 `rmdir()` の使用例

[「chdir」の使用例」\(p320\)](#) を参照してください。

sleep

指定された秒数、プログラムの実行を遅らせます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unistd.h>`
`unsigned int sleep(unsigned int sleep);`

引数 この関数の引数です。

sleep unsigned int 秒単位の時間

注意点 関数 `sleep()` は、`unsigned` の整数型の引数で指定された時間、プログラムの実行を遅らせます。Mac OS システムでは、返すエラーの値はありません。使用例を [「sleep\(\) の使用例」\(p334\)](#) に示します。

戻り値 sleep() はゼロを返します。

例 29.12 sleep() の使用例

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{

    printf("Output to window\n");
    fflush(stdout); /* 出力させる必要あり */

    sleep(3);

    printf("Second output to window");

    return 0;
}
```

Result
Output to window
< there is a delay now >
Second output to window

ttyname

ファイル ID に対応した端末の名前を検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <unistd.h>
char *ttyname(int fildes);
```

引数 この関数の引数です。

fildes	int	ファイルの説明
--------	-----	---------

注意点 関数 ttyname() は、ファイル ID に対応した端末の名前を検索します。

戻り値 ttyname() は、ファイル ID に対応した端末の名前を指す文字型のポインタを返します。ID が端末を指定していない場合、NULL を返します。

例 29.13 ttynam() の使用例

[「isatty\(\)、ttynam\(\) の使用例」\(p330\)](#) を参照してください。

unlink

ファイルを削除（アンリンク）します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <unistd.h>
int unlink(const char *path);
```

引数 この関数の引数です。

path const char * 削除するファイルのパス名

注意点 関数 unlink() はディレクトリから指定されたファイルを削除します。使用例を[「unlink\(\) の使用例」\(p335\)](#) に示します。

戻り値 成功した場合、unlink() はゼロを返します。失敗した場合、負の値を返します。

参照 [「rmdir」\(p333\)](#)、[「mkdir」\(p160\)](#)

例 29.14 unlink() の使用例

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define SIZE FILENAME_MAX

int main(void)
{
    FILE *fp;
    char fname[SIZE] = "Test.txt";

    /* ファイルの生成 */
    if( (fp =fopen( fname,"w" ) ) == NULL )
    {
        printf("Can not open %s for writing", fname);
        exit( EXIT_FAILURE);
    }
    else printf("%s was opened for writing\n",fname);
```

```
/* ファイルが利用可能であることを表示 */
if( !fclose(fp) ) printf("%s was closed\n",fname);

/* ファイルの削除 */
if( unlink(fname) )
{
    printf("%s was not deleted",fname);
    exit( EXIT_FAILURE );
}

/* 再オープンできないことを示す */
if( (fp =fopen( fname,"r" ) ) == NULL )
{
    printf("Can not open %s for reading it was deleted",
          fname);
    exit( EXIT_FAILURE);
}
else printf("%s was opened for reading\n",fname);

return 0;
}
```

Result
Test.txt was opened for writing
Test.txt was closed
Can not open Test.txt for reading it was deleted

write

書式化されていないファイルストリームへ書き込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unistd.h>`
`int write(int fildes, const char *buf, int count);`

引数 この関数の引数です。

fildes	int	ファイルの説明
buf	const char *	書き込まれているバッファのアドレス
count	int	書き込まれているバッファのサイズ

注意点 関数 `write()` は、ファイル `fildes` に文字配列バッファから引数 `count` バイト分をコピーします。

ファイル位置は、書き込まれたバイト数分増加されます。

注意： この関数は、[「read」\(p332 \)](#) と [「open」\(p56 \)](#) のみの組み合わせで利用すべきです。

戻り値 `write()` は、実際に書き込まれたバイト数を返します。

参照 [「fwrite」\(p205 \)](#)、[「read」\(p332 \)](#)、[「open」\(p56 \)](#)

例 29.15 `write()` の使用例

[「close\(\) の使用例」\(p322 \)](#) を参照してください。

第 30 章 unix.h

unix.h ヘッダファイルには、UNIX からプログラムを移植するための便利な関数があります。

unix.h の概要

unix.h ヘッダファイルの関数は多くの UNIX ライブラリの関数と類似しています。しかし UNIX と Mac OS のオペレーティングシステムは基本的に異なり、同じではありません。関数の説明で何が違うのかを示します。

unix.h の大域変数と機能は以下に示します。

[「_fcreator」\(p339\)](#) : Mac OS ファイルクリエータを設定する

[「_ftype」\(p340\)](#) : Mac OS のファイルタイプを設定する

[「fdopen」\(p341\)](#) : ファイル記述子をストリームに変換する

[「fileno」\(p342\)](#) : ストリームをファイル記述子に変換する

[「tell」\(p343\)](#) : ファイルのオフセットを決定する

unix.h と UNIX の互換性

一般的に新しいプログラムではこれらの関数を使わず、代わりに API 本来の代用プログラムを使います。

注意 : UNIX あるいは DOS のプログラムを移植しているのである場合、他の UNIX の互換ヘッダの関数が必要かもしれません。

大域変数

新しいファイルのタイプとクリエータを設定する大域変数です。

_fcreator

Mac OS のファイルクリエータを指定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unix.h>`
`extern long _fcreator`

注意点 大域変数 `_fcreator` は、標準の C のライブラリを使って生成されたファイルに関するクリエータのタイプを設定する場合に使用されます。[「ファイルクリエータとタイプの設定をするために大域変数を利用」\(p340\)](#) は大域変数 variable `_fcreator` の使用例です。

`_ftype`

Mac OS のファイルのタイプを指定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unix.h>`
`extern long _ftype;`

注意点 大域変数 `_ftype` は、標準の C のライブラリを使って生成されたファイルに関するクリエータの型を設定する場合に使用されます。[「ファイルクリエータとタイプの設定をするために大域変数を利用」\(p340\)](#) は、大域変数 variable `_fcreator` の使用例です。

ヒント： `_fcreate` と `_ftype` に割り当てられた値は、ResType です（4 文字の定数）。

例 30.1 ファイルクリエータとタイプの設定をするために大域変数を利用

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unix.h>

#define oFile "test file"
const char *str = "Metrowerks Software at Work";

int main(void)
{
    FILE *fp;
    _fcreator = 'ttxxt';
    _ftype = 'TEXT';

    // 入出力用に新しいファイルを生成
    if (( fp = fopen(oFile, "w+")) == NULL)
    {
        printf("Can't create file.\n");
    }
}
```

```

        exit(1);
    }

    fwrite(str, sizeof(char), strlen(str), fp);
    fclose(fp);

    return 0;
}

```

// fwrite() を使ってファイルへ出力
Metrowerks Software at Work

fdopen

ファイル記述子をストリームへ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <unix.h>
FILE *fdopen(int fildes, char *mode);

引数 この関数の引数です。

fildes	int	ファイルの説明
mode	char *	ファイルのオープンモード

注意点 この関数は、ファイル記述子 `fildes` に対するストリームを生成します。`fprintf()` と `getchar()` の標準の I/O 関数を使って、ストリームを利用することができます。CodeWarrior C/C++ では、引数 `mode` の値は無視されます。

戻り値 成功した場合、`fdopen()` はストリームを返します。エラーが発生した場合、`fdopen()` は `NULL` を返します。

参照 [「fileno」\(p342\)](#)、[「open」\(p56\)](#)

例 30.2 fdopen() の使用例

```

#include <stdio.h>
#include <unix.h>

int main(void)
{
    int fd;

```

```
FILE *str;

fd = open("mytest", O_WRONLY | O_CREAT);

/* ファイル記述子へ書き込み */
write(fd, "Hello world!\n", 13);
/* ファイル記述子からストリームへ変換 */

str = fdopen(fd, "w");

/* ストリームへ書き込み */
fprintf(str, "My name is %s.\n", getlogin());

/* ストリームをクローズ */
fclose(str);
/* ファイル記述子をクローズ */
close(fd);

return 0;
}
```

fileno

ストリームからファイル記述子へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <unix.h>
int fileno(FILE *stream);

引数 この関数の引数です。

stream FILE * FILE ストリームへのポインタ

注意点 この関数は、ファイル記述子 `filides` に対するストリームを生成します。unix.h 中の他の関数、例えば、`read()` と `write()` と共に、ファイル記述子を利用することができます。

標準 I/O ストリーム `stdin`、`stdout`、`stderr` に関しては、`fileno()` は以下の値を返します。

表 30.1 標準 I/O ストリームに関するファイル記述子

関数の呼び出し ...	ファイル記述子への戻り値 ...
<code>fileno(stdin)</code>	0
<code>fileno(stdout)</code>	1
<code>fileno(stderr)</code>	2

戻り値 成功した場合、`fileno()` はファイル記述子を返します。エラーの場合、-1 を返し `errno` に値を設定します。

参照 [「fdopen」\(p341\)](#)、[「open」\(p56\)](#)

例 30.3 `fdopen()` の使用例

```
#include <unix.h>

int main(void)
{
    write(fileno(stdout), "Hello world!\n", 13);

    return 0;
}
```

```
Reult
Access time:      Tue Apr 18 22:28:22 1995
Modification time: Tue Apr 18 22:28:22 1995
Creation time:    Tue Apr 18 11:28:41 1995
Block size:      11264
Number of blocks: 1
```

tell

ファイルに関する現在のオフセットを返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <unix.h>`
`long tell(int fildes);`

引数 この関数の引数です。

`fildes` `int` ファイルの説明 `s`

注意点 この関数はファイル記述子に関連するファイルの現在のオフセットを返します。値はファイルの始めからのバイト数です。

戻り値 成功した場合、`tell()` はオフセットを返します。エラーの場合、`tell()` は-1Lを返します。

参照 [「ftell」\(p204\)](#)、[「lseek」\(p331\)](#)

例 30.4 read() の使用例

```
#include <stdio.h>
#include <unix.h>

int main(void)
{
    int fd;
    long int pos;

    fd = open("mytest", O_RDWR | O_CREAT | O_TRUNC);
    write(fd, "Hello world!\n", 13);
    write(fd, "How are you doing?\n", 19);

    pos = tell(fd);

    printf("You're at position %ld.", pos);

    close(fd);

    return 0;
}
```

Result

This program prints the following to standard output:
You're at position 32.

第 31 章 utime.h

utime.h ヘッダファイルには、UNIX からプログラムを移植するためのいくつかの使い易い関数が含まれています。

utime.h の概要

utime.h ヘッダファイルの関数は多くの UNIX ライブラリの関数と同様です。しかし UNIX と MacOS のオペレーティングシステムは基本的に異なる部分があり、同じにはなりません。以下の節でこの違いについて説明します。

[「utime」\(p345\)](#)：ファイルの変更時間を設定する

[「utimes」\(p347\)](#)：ファイルの変更時間の一組を設定する

utime.h と UNIX の互換性

一般的に新しいプログラムではこれらの関数を使わず、代わりに API 本来の対応するものを利用します。

注意： UNIX や DOS のプログラムを移植するのである場合、UNIX と互換のある他のヘッダの関数が必要になるかもしれません。

utime

ファイルの変更時間を設定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <utime.h>
int utime(const char *path,
          const struct utimbuf *buf);
```

引数 この関数の引数です。

path	const char *	パス名の文字列
buf	const struct utimbuf *	ファイルの時間情報を持つ構造体のアドレス

注意点 この関数は、path で指定されたファイルに関する変更時間を設定します。Mac OS はファイルアクセス時間と一致するものを持たないので、utimbuf 構造体の actime フィールドは無視されます。

buf がNULLである場合、utime() は変更時間を現在時間に設定します。buf がutimbuf構造体を指していれば、utime() は変更時間を構造体の modtime フィールドに指定された時間に設定します。

utimbuf 構造体には[表 31.2](#) に示すフィールドがあります。

表 31.1 utimbuf 構造体

フィールド		フィールド
time_t	actime	ファイルへのアクセス時間。Mac OS は、これに対応するものがないので、utime() はこのフィールドを無視する。
time_t	modtime	ファイルが変更された最後の時間。

戻り値 成功した場合、utime() はゼロを返します。エラーの場合、utime() は -1 を返し errno を設定します。

参照 [「utimes」\(p347\)](#)、[「ctime」\(p307\)](#)、[「mktime」\(p310\)](#)、[「stat」\(p161\)](#)、[「fstat」\(p158\)](#)

例 31.1 utime() の例

```
#include <stdio.h>
#include <unix.h>

int main(void)
{
    struct utimbuf timebuf;
    struct tm date;
    struct stat info;

    /* カレンダー時間の生成
    1994 年 4 月 4 日午前 0 時 */
    date.tm_sec=0; /* 0 秒 */
    date.tm_min=0; /* 0 分 */
    date.tm_hour=0; /* 0 時 */
    date.tm_mday=4; /* 4 日 */
    date.tm_mon=3; /* .. 4 月 */
    date.tm_year=64; /* ...1994 年 */
    date.tm_isdst=-1; /* 夏時間ではない */
    timebuf.modtime=mktime(&date);
    /* カレンダー時間に変換 */
}
```

```

/* 変更日付を 1964 年 4 月 4 日  *
 * 午前 0 時へ変更 */
utime("mytest", &timebuf);
stat("mytest", &info);
printf("Mod date is %s", ctime(&info.st_mtime));

/* 変更日付を現在に変更 */
 * right now.                */
utime("mytest", NULL);
stat("mytest", &info);
printf("Mod date is %s", ctime(&info.st_mtime));

return 0;
}

```

This program might print the following to standard output:
Mod date is Sat Apr 4 00:00:00 1964
Mod date is Mon Apr 10 20:43:09 1995

utimes

ファイルの変更時間を設定します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <utime.h>`
`int utimes(const char *path,`
`struct timeval buf[2]);`

引数 この関数の引数です。

<code>path</code>	<code>const char *</code>	パス名の文字列
<code>buf</code>	<code>timeva struct array</code>	修正日時の設定に使われた時間の値の配列

注意点 この関数は、`path` で指定されたファイルに関する変更時間を配列 `buf` の 2 番目の要素に設定します。配列 `buf` のそれぞれの要素は、[表 31.2](#) のフィールドを持つ `timeval` 構造体です。

表 31.2 timeval 構造体

フィールド		フィールド
<code>int t</code>	<code>tv_sec</code>	秒。
<code>int</code>	<code>tv_usec</code>	マイクロ秒。Mac OS は、マイクロ秒を使わないので <code>utimes()</code> はこれを無視する。

buf の 1 番目の要素はアクセス時間です。

注意： Mac OS は、ファイルアクセス時間に一致するものを持たないので、配列のその要素は無視されます。

戻り値 成功した場合、utimes() は 0 を返します。エラーの場合、utimes() は -1 を返し errno を設定します。

参照 [「utime」\(p345\)](#)、[「ctime」\(p307\)](#)、[「mktime」\(p310\)](#)、[「fstat」\(p158\)](#)、[「stat」\(p161\)](#)

例 31.2 utimes() の例

```
#include <stdio.h>
#include <unix.h>

int main(void)
{
    struct tm date;
    struct timeval buf[2];
    struct stat info;

    /* カレンダー時間の生成
    1965 年 9 月 9 日午前 0 時 */
    date.tm_sec=0;      /* 0 秒 */
    date.tm_min=0;      /* 0 分 */
    date.tm_hour=0;     /* 0 時 */
    date.tm_mday=9;     /* 9 日 */
    date.tm_mon=8;      /* 9 月 */
    date.tm_year=65;    /* 1965 年 */
    date.tm_isdst=-1;   /* 夏時間ではない */
    buf[1].tv_sec=mktime(&date);
    /* カレンダー時間に変更 */

    /* 変更日付を 1965 年 9 月 9 日午前 0 時に変更 */
    utimes("mytest", buf);
    stat("mytest", &info);
    printf("Mod date is %s", ctime(&info.st_mtime));

    return 0;
}
```

This program prints the following to standard output:
Mod date is Thu Sep 9 00:00:00 1965

第 32 章 utsnane.h

ヘッダファイル `utsnane.h` には、UNIX からプログラムを移植するための便利な関数が含まれています。

utsnane.h の概要

`utsnane.h` ヘッダファイルの関数は多くの UNIX ライブラリの関数と類似しています。しかし UNIX と Mac OS オペレーティングシステムは基本的に異なり、同じではありません。以下の節では、違いについて説明します。

[「uname」\(p349\)](#)：現在使用中のシステムの情報を検索する

utsnane.h と UNIX の互換性

一般的に新しいプログラムにこれらの関数を使わず、代わりに API 本来の代用プログラムを使います。

注意： UNIX あるいは DOS のプログラムを移植している場合、他の UNIX の互換ヘッダの関数が必要かもしれません。

uname

現在使用中のシステムに関する情報を取得します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <utsnane.h>
int uname(struct utsnane *name);
```

引数 この関数の引数です。

name struct utsnane * システム情報を保存する構造体

注意点 この関数は現在使用中の Mac OS 上での情報を得て、その情報を `name` で指定された構造体へ格納します。この構造体は[表 32.1](#) で列挙されたフィールドを持っています。全てのフィールドはヌルで終わる文字列です。

表 32.1 utsname 構造体

フィールド	説明
sysnam	オペレーティングシステム
nodename	共有 ノード名
release	システムソフトウェアのリリース番号
version	システムソフトウェアバージョンのマイナーリリース番号
machine	使用しているマシンの型

戻り値 成功した場合、`uname()` はゼロを返します。エラーの場合、`uname()` は -1 を返し `errno` を設定します。

参照 [「fstat」\(p158\)](#)、[「stat」\(p161\)](#)

例 32.1 uname() の使用例

```
#include <stdio.h>
#include <utsname.h>

int main(void)
{
    struct utsname name;

    uname(&name);

    printf("Operating System: %s\n", name.sysname);
    printf("Node Name:           %s\n", name.nodename);
    printf("Release:                %s\n", name.release);
    printf("Version:                 %s\n", name.version);
    printf("Machine:                 %s\n", name.machine);

    return 0;
}
```

This application could print the following:
Operating System: MacOS
Node Name: Chan's PowerMac
Release: 7
Version: 51
Machine: Power Macintosh
This machine is a Power Macintosh running Version 7.5.1 of the
MacOS. The Macintosh Name field of the Sharing Setup control panel
contains "Chan's PowerMac"

第 33 章 wchar.h

wchar.h ヘッダファイルには拡張文字セットを扱うための、さまざまな定義や関数宣言があります。

wchar.h の概要

wchar.h ヘッダファイルの以下の関数について説明します。

入出力機能

[「fgetwc」](#) : 拡張文字用の fgetc 関数

[「fgetws」](#) : 拡張文字用の fgets 関数

[「fputwc」](#) : 拡張文字用の fputc 関数

[「fputws」](#) : 拡張文字用の fputs 関数

[「fwprintf」](#) : 拡張文字用の fprintf 関数

[「fwscanf」](#) : 拡張文字用の fscanf 関数

[「getwc」](#) : 拡張文字用のgetc 関数

[「getwchar」](#) : 拡張文字用の getchar 関数

[「putwc」](#) : 拡張文字用のputc 関数

[「putwchar」](#) : 拡張文字用の putchar 関数

[「swprintf」](#) : 拡張文字用の sprintf 関数

[「swscanf」](#) : 拡張文字用の sscanf 関数

[「wprintf」](#) : 拡張文字用の printf 関数

[「wscanf」](#) : 拡張文字用の scanf 関数

[「vwprintf」](#) : 拡張文字用の vfprintf 関数

[「__vwscanf」](#) : 拡張文字用の vfscanf 関数

[「__vswscanf」](#) : 拡張文字用の vsscanf 関数

[「vprintf」](#) : 拡張文字用の vprintf 関数

[「vswprintf」](#) : 拡張文字用の vsprintf 関数

[「vwscanf」](#) : 拡張文字用の vscanf 関数

タイム機能

[「wasctime」](#) : 拡張文字用の asctime 関数

[「wcsftime」](#) : 拡張文字用の csftime 関数

[「wctime」](#) : 拡張文字用の ctime 関数

マッピング機能

[「towctrans」](#) : 大小文字と拡張文字のマッピング関数

[「wctrans」](#) : 拡張文字用のマッピング関数

文字列機能

[「watof」](#) : 拡張文字用の atof 関数

[「wcscat」](#) : 拡張文字用の strcat 関数

[「wcschr」](#) : 拡張文字用の strchr 関数

[「wcscmp」](#) : 拡張文字用の strcmp 関数

[「wcscspn」](#) : 拡張文字用の strspn 関数

[「wcscoll」](#) : 拡張文字用の strcoll 関数

[「wcscpy」](#) : 拡張文字用の strcpy 関数

[「wcslen」](#) : 拡張文字用の strlen 関数

[「wcsncat」](#) : 拡張文字用の strncat 関数

[「wcsncmp」](#) : 拡張文字用の strncmp 関数

[「wcsncpy」](#) : 拡張文字用の strncpy 関数

[「wcspbrk」](#) : 拡張文字用の strbrk 関数

[「wcstod」](#) : 拡張文字用の strtod 関数

[「wcsrchr」](#) : 拡張文字用の strrchr 関数

[「wcssp」](#) : 拡張文字用の strspn 関数

[「wcsstr」](#) : 拡張文字用の strstr 関数

[「wcstok」](#) : 拡張文字用の strtok 関数

[「wcxfrm」](#) : 拡張文字用の strxfrm 関数

[「wmemcpy」](#) : 拡張文字用の memcpy 関数

[「wmemmove」](#) : 拡張文字用の memmove 関数

[「wmemset」](#) : 拡張文字用の memset 関数

[「wmemchr」](#) : 拡張文字用の memchr 関数

[「wmemcmp」](#) : 拡張文字用の memcmp 関数

拡張文字とバイト文字のストリーム方向

入力と出力には 2 種類のストリーム方向があります。拡張文字 (wchar_t) 方向とバイト (char) 方向です。ストリームがファイルと関連づけられた後は、ある操作が行われるまでストリームに方向はなくなります。

ストリームで何らかの操作が行われると、その操作によってストリームに拡張文字またはバイトの方向が生じます。その方向はファイルを閉じて再度開くまで残ります。

ストリームの方向が確立された後、他の方向の関数呼び出しはできません。つまりバイト方向の I/O 関数は、拡張文字方向のストリームでは効果がありません。

ストリーム方向と標準の入力 / 出力

あらかじめ定義されている 3 つのストリーム (stdin、stdout、stderr) は、プログラムの起動時には方向が決まっていません。標準の入力 / 出力ストリームが閉じていると、コンソールへのそのストリームを再度開いて接続することはできません。しかし、名前のあるファイルへのストリームを再度開いて接続することはできます。

C/C++ の入力 / 出力関数は同じ stdin、stdout、stderr ストリームを共有します。

定義

ヘッダファイル `wchar.h` には拡張文字セットを使うための特別な定義が含まれています。

表 33.1 拡張文字の定義

定義	内容
WCHAR_MIN	拡張文字の最小サイズ
WCHAR_MAX	拡張文字の最大サイズ
WEOF	ファイルの最後
EILSEQ	拡張文字のシーケンスエラー
wctrans_t	特別の拡張文字マッピングに対するスカラー型

fgetwc

ファイルストリームから拡張文字を 1 文字取得します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
wchar_t fgetwc(FILE * file);
```

引数 この関数の引数を以下に示します。

file FILE * 文字を取得するファイル

注意点 拡張文字用に fgetc 関数と同様の処理を行います。

戻り値 文字を返す、またはエラーの場合は WEOF を返します。

注意： embedded/RTOS システム上では、この関数は stdin、stdout、stderr ファイルにのみ実装されています。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「fgetc」\(p180\)](#)

fgetws

関数 fgetws はファイルストリームから 1 つの拡張文字列を読み出します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ #include <wchar.h>
wchar_t *fgetws(wchar_t * s, int n, FILE * file);

引数 この関数の引数を以下に示します。

s	wchar_t *	入力用の拡張文字文字列
n	int	拡張文字の個数
file	FILE *	ファイルストリームへのポインタ

注意点 拡張文字用の fgets 関数です。

注意： embedded/RTOS システム上では、この関数は stdin、stdout、stderr ファイルにのみ実装されています。

戻り値 成功した場合、s へのポインタを返します。失敗した場合、EOF や NULL を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「fgets」\(p183\)](#)

fwprintf

フォーマットしてファイルへ挿入します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`int fwprintf(FILE * file,`
`const wchar_t * format, ...);`

引数 この関数の引数を以下に示します。

file	FILE *	ファイルストリームへのポインタ
format	wchar_t *	フォーマットを定義する文字列
....		可変個の引数

注意点 拡張文字用の fprintf 関数です。

fwprintf() 関数はフォーマットに沿ってテキストをストリームへ書き込み、ファイル位置を進めます。動作は wprintf() と同じでそれにストリーム引数を追加しただけです。printf() の定義を参照してください。

注意： embedded/RTOS システム上では、この関数は stdin、stdout、stderr ファイルにのみ実装されています。

戻り値 書き込みに成功した引数の個数を返します。エラーの場合、負の数を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「fprintf」\(p187\)](#)

fputwc

ファイルストリームに 1 つのシングル拡張文字を挿入します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`wchar_t fputwc(wchar_t c, FILE * file);`

引数 この関数の引数を以下に示します。

c	wchar_t	挿入する文字
file	FILE *	ファイルストリームへのポインタ

注意点 拡張文字用に fputc 関数と同様の処理を行います。

注意： embedded/RTOS システム上では、この関数は stdin、stdout、stderr ファイルにのみ実装されています。

戻り値 成功した場合、書き込んだ文字を返します。失敗した場合、WEOF を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「fputc」\(p191\)](#)

fputws

ファイルストリームへ拡張文字列を挿入します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
int fputws(const wchar_t * s, FILE * file);
```

引数 この関数の引数を以下に示します。

s	wchar_t *	挿入する文字列
file	FILE *	ファイルストリームへのポインタ

注意点 拡張文字用に fputs 関数と同様の処理を行います。

ファイルが更新モード(+)で開かれた場合、ストリームバッファをフラッシュする操作によって書き出しと読み込み操作が分離されない限りファイルへ書き出したり、ファイルから読み込んだりすることができません。この操作は fflush() 関数またはファイル位置を変更する関数のいずれか(fseek()、fsetpos()、rewind())と共に使う必要があります。

注意： embedded/RTOS システム上では、この関数は stdin、stdout、stderr ファイルにのみ実装されています。

戻り値 成功した場合、ゼロを返します。失敗した場合、非ゼロの値を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「fputs」\(p193\)](#)

fwscanf

ストリームからフォーマットしたテキストを読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`int fwscanf(FILE * file,`
`const wchar_t * format, ...);`

引数 この関数の引数を以下に示します。

file	FILE *	ファイルストリーム
format	wchar_t *	フォーマットを定義する文字列
....		可変個の引数

注意点 拡張文字用に `fscanf` 関数と同様の処理を行います。

`fwscanf()` 関数はプログラマが定義したフォーマットでストリームからテキストを読み込みます。この関数は `wscanf()` 関数とほぼ同様の処理を行い、読み込むストリームが引数に追加されています。 `scanf()` 関数の定義を参照してください。

注意： `embedded/RTOS` システム上では、この関数は `stdin`、`stdout`、`tderr` ファイルにのみ実装されています。

戻り値 読み込んだ項目数を返します。フォーマットを定義する文字列と一致しない読み込みエラーが発生した場合、`fwscanf()` は `errno` に非ゼロの値を定義します。ファイルの終わりに達した場合、`fwscanf()` は `WEOF` を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「fscanf」\(p197\)](#)

getwc

ファイルストリームから読み込んだ拡張文字型を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`wchar_t getwc(FILE * file);`

引数 この関数の引数を以下に示します。

file	FILE *	ファイルストリーム
------	--------	-----------

注意点 拡張文字用に `getc` 関数と同様の処理を行います。

戻り値 ストリームから次の文字を返します。ファイルの終わり、またはエラーの場合 WEOF を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353 \)](#) [「getc」\(p206 \)](#)

getwchar

標準入力からの拡張文字型を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
wchar_t getwchar(void);
```

引数 なし。

注意点 拡張文字用に getchar 関数と同様の処理を行います。

戻り値 成功した場合、stdin から int 型で次の文字の値を返します。ファイルの終わり、またはエラーの場合 WEOF を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353 \)](#) [「getwchar」\(p358 \)](#)

putwc

ストリームへ拡張文字型の文字を書き出します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
wchar_t putwc(wchar_t c, FILE * file);
```

引数 この関数の引数を以下に示します。

c	wchar_t	書き出す文字
file	FILE	出力ストリーム

注意点 拡張文字用に putc 関数と同様の処理を行います。

戻り値 成功した場合、書き出した文字 c を返します。失敗した場合、WEOF を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「putc」\(p217\)](#)

putwchar

標準出力へ拡張文字型の文字を書き出します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`wchar_t putwchar(wchar_t c);`

引数 この関数の引数を以下に示します。

c	wchar_t	書き出す拡張文字型の文字
---	---------	--------------

注意点 拡張文字用に `putchar` 関数と同様の処理を行います。

戻り値 成功した場合、書き出した文字 `c` を返します。失敗した場合、`WEOF` を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「putchar」\(p218\)](#)

swprintf

拡張文字型文字列をフォーマットします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`int swprintf(wchar_t * S, size_t N,`
`const wchar_t * format, ...);`

引数 この関数の引数を以下に示します。

s	wchar_t*	書式化したテキストを保存する文字列バッファ
n	size_t	書き込みを許可された文字の数
format	wchar_t*	フォーマットを定義する文字列
....		変数の引数

注意点 拡張文字が書き込み可能な数に追加パラメータがある場合、拡張文字用に `sprintf` 関数と同様の処理を行います。終末の `NULL` 拡張文字 (`n` がゼロでない限り追加される) を含む `n` 以上の拡張文字は書き込みできません。

戻り値 `s` にアサインされた文字数 (ヌル文字以外) を返します。または `N` や他の文字を書き込む場合は負の数値を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「fwprintf」\(p354\)](#)、[「sprintf」\(p231\)](#)

swscanf

フォーマットして拡張文字型文字列を読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
int swscanf(const wchar_t * s,
            const wchar_t * format, ...);
```

引数 この関数の引数を以下に示します。

<code>s</code>	<code>wchar_t*</code>	読み込む文字列
<code>format</code>	<code>wchar_t*</code>	フォーマットを定義する文字列
<code>...</code>		可変個の引数

注意点 拡張文字用に `sscanf` 関数と同様の処理を行います。

戻り値 読み込み / 変換に成功した項目数を返します。ファイルの終わり、または変換仕様が引数と一致しない場合、`WEOF` を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「sscanf」\(p232\)](#)

towctrans

拡張文字型を他の拡張文字型にマッピングします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
wint_t towctrans(wint_t c, wctrans_t value);
```

引数 この関数の引数を以下に示します。

<code>c</code>	<code>wint_t</code>	マップする文字
<code>value</code>	<code>wctrans_t</code>	マップの方法を示す値

注意点 最初の引数を value で指定された大文字または小文字の値へマップします。

戻り値 マップされた文字を返します。

参照 [「wctrans」\(p375\)](#)

__vfwscanf

フォーマットしてファイルから読む込んだ的可変引数を求めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
int __vfwscanf(FILE * file,
               const wchar_t * format_str, va_list arg);
```

引数 この関数の引数を以下に示します。

file	FILE *	読み込むストリーム
format_str	const wchar_t*	フォーマットを定義する文字列
....		可変個の引数

注意点 拡張文字用に fscanf 関数と同様の処理を行います。

注意： embedded/RTOS システム上では、この関数は stdin、stdout、tderr ファイルにのみ実装されています。

戻り値 成功した場合、スキャンした項目数を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「fscanf」\(p197\)](#)

__vswscanf

フォーマットして文字列から読み込んだ可変引数を求めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
int __vswscanf(const wchar_t * s,
               const wchar_t * format, va_list arg);
```

引数 この関数の引数を以下に示します。

s	wchar_t*	読み取りされている文字列
format	wchar_t*	フォーマットを定義する文字列
....		可変個の引数

注意点 拡張文字用に `sscanf` 関数と同様の処理を行います。

注意： `embedded/RTOS` システム上では、この関数は `stdin`、`stdout`、`stderr` ファイルにのみ実装されています。

戻り値 成功した場合、スキャンした項目数を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「sscanf」\(p232\)](#)

vwscanf

フォーマットした文字列を読み込むための可変引数を求めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
int vwscanf(const wchar_t * format, va_list arg);
```

引数 この関数の引数を以下に示します。

s	wchar_t*	読み込む引数
format	wchar_t*	フォーマットを定義する文字列
....		可変個の引数

注意点 拡張文字用に `sscanf` 関数と同様の処理を行います。

戻り値 成功した場合、スキャンした項目数を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「__vwscanf」\(p361\)](#)、[「scanf」\(p224\)](#)

vfwprintf

ファイルストリームへフォーマットしたテキストを書き出します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`int vfwprintf(FILE * file,`
`const wchar_t * format_str, va_list arg);`

引数 この関数の引数を以下に示します。

file	FILE *	書き出し先のストリーム
format_str	wchar_t*	フォーマットを定義する文字列
....		可変個の引数

注意点 拡張文字用に `vfpirntf` 関数と同様の処理を行います。

注意： embedded/RTOS システム上では、この関数は `stdin`、`stdout`、`tderr` ファイルにのみ実装されています。

戻り値 書き出した文字数を返します。失敗した場合、`WEOF` を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「vfprintf」\(p237\)](#)

vswprintf

文字列へフォーマットされた出力を書き出します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`int vswprintf(wchar_t * s,`
`const wchar_t * format, va_list arg);`

引数 この関数の引数を以下に示します。

s	wchar_t*	読み込む文字列
format	wchar_t*	フォーマットを定義する文字列
....		可変個の引数

注意点 拡張文字用に `vsprintf` 関数と同様の処理を行います。

戻り値 書き出した文字数を返します。失敗した場合、`WEOF` を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#) [「vsprintf」\(p240\)](#)

vwprintf

stdout へ可変個の引数をフォーマットした出力を書き出します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
int vwprintf(const wchar_t * format, va_list arg);
```

引数 この関数の引数を以下に示します。

format	wchar_t*	フォーマットを定義する文字列
....		可変個の引数

注意点 拡張文字用に vprintf 関数と同様の処理を行います。

戻り値 書き出した文字数を返します。失敗した場合、負の値を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#) [「vprintf」\(p239\)](#)

wasctime

tm 構造体を拡張文字配列へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
wchar_t * wasctime(const struct tm * tm);
```

引数 この関数の引数を以下に示します。

tm	const struct tm *	タイム値を持つ構造体
----	-------------------	------------

注意点 拡張文字用に asctime 関数と同様の処理を行います。

戻り値 変換した tm 構造体を含むヌル文字終了の配列へのポインタを返します。

参照 [「asctime」\(p305\)](#)

watof

拡張文字型文字列を double 型へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`double watof(wchar_t * str);`

引数 この関数の引数を以下に示します。

x double 変換する文字列

注意点 拡張文字用に atof 関数と同様の処理を行います。

戻り値 変換した値を返します。

参照 [「atof」\(p248\)](#)

wscat

拡張文字型文字列を後へ追加します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`wchar_t * wscat(wchar_t * dst,`
`const wchar_t * src);`

引数 この関数の引数を以下に示します。

dst wchar_t * 追加先の文字列
src wchar_t * ソース文字列

注意点 拡張文字用に strcat 関数と同様の処理を行います。

戻り値 追加先の文字列へのポインタを返します。

参照 [「strcat」\(p281\)](#)

wcschr

拡張文字を検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
wchar_t * wcschr(const wchar_t * str,
                 const wchar_t chr);
```

引数 この関数の引数を以下に示します。

str	wchar_t	検索される文字列
chr	wchar_t	検索する文字

注意点 拡張文字用に `strchr` 関数と同様の処理を行います。

戻り値 成功した場合、その文字へのポインタを返します。失敗した場合、ヌルポインタを返します。

参照 [「strchr」\(p282\)](#)

wcscmp

2 つの拡張文字型配列を比較します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
int wcscmp(const wchar_t * str1,
           const wchar_t * str2);
```

引数 この関数の引数を以下に示します。

str1	wchar_t*	比較文字列その 1
str2	wchar_t*	比較文字列その 2

注意点 拡張文字用に `strcmp` 関数と同様の処理を行います。

戻り値 `str1` と `str2` が等しい場合はゼロを返します。`str1` が `str2` より小さい場合は負の値を返し、`str1` が `str2` より大きい場合は正の値を返します。

参照 [「strcmp」\(p283\)](#)、[「wmemcmp」\(p376\)](#)

wscoll

2 つの拡張文字型配列を比較します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
int wscoll(const wchar_t *str1,
           const wchar_t * str2);
```

引数 この関数の引数を以下に示します。

str1	wchar_t *	比較文字列その 1
str2	wchar_t *	比較文字列その 2

注意点 拡張文字用に strcoll 関数と同様の処理を行います。

戻り値 str1 と str2 が等しい場合はゼロを返します。str1 が str2 より小さい場合は負の値を返し、str1 が str2 より大きい場合は正の値を返します。

参照 [「strcoll」\(p284 \)](#)、[「wcscmp」\(p366 \)](#)、[「wmemcmp」\(p376 \)](#)

wscspn

拡張文字配列において、他の拡張文字セットにはない文字の数を数えます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
size_t wscspn(const wchar_t * str,
              const wchar_t * set);
```

引数 この関数の引数を以下に示します。

str	wchar_t *	検索される文字列
set	wchar_t *	検索する拡張文字のセット

注意点 拡張文字用に strcspn 関数と同様の処理を行います。

戻り値 set のどの文字とも一致しない、str 中に含まれる文字の数を返します。

参照 [「strcspn」\(p286 \)](#)

wcscpy

拡張文字配列をコピーします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
wchar_t * (wcscpy)(wchar_t * dst,
    const wchar_t * src);
```

引数 この関数の引数を以下に示します。

dst	wchar_t *	コピー先の文字列
src	wchar_t *	コピー元の文字列

注意点 拡張文字用に strcpy 関数と同様の処理を行います。

戻り値 コピー先の文字列へのポインタを返します。

参照 「[strcpy](#)」(p285)

wcslen

拡張文字配列の長さを計算します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
size_t (wcslen)(const wchar_t * str);
```

引数 この関数の引数を以下に示します。

str	wchar_t *	計算する文字列
-----	-----------	---------

注意点 拡張文字用に strlen 関数と同様の処理を行います。

戻り値 文字配列の文字数（終端ヌル文字は数えない）を返します。

参照 「[strlen](#)」(p289)

wcsncat

指定された文字数の文字を拡張文字配列へ追加します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`wchar_t * wcsncat(wchar_t * dst,`
`const wchar_t * src, size_t n);`

引数 この関数の引数を以下に示します。

dst	wchar_t *	コピー先の文字列
src	wchar_t *	追加する文字列
n	size_t	追加する文字数

注意点 拡張文字用に `strncat` 関数と同様の処理を行います。

戻り値 コピー先の文字列へのポインタを返します。

参照 「[strncat](#)」(p290)

wcsncmp

指定された文字数の拡張文字を比較します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`int wcsncmp(const wchar_t * str1,`
`const wchar_t * str2, size_t n);`

引数 この関数の引数を以下に示します。

str1	wchar_t *	比較文字列その 1
str2	wchar_t *	比較文字列その 2
n	size_t	比較する文字数

注意点 拡張文字用に `strncmp` 関数と同様の処理を行います。

戻り値 `str1` と `str2` 最初の `n` 文字が等しい場合はゼロを返します。`str1` が `str2` より小さい場合は負の値を返し、`str1` が `str2` より大きい場合は正の値を返します。

参照 [「strncmp」\(p291 \)](#)

wcsncpy

指定された文字数の拡張文字をコピーします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
wchar_t * wcsncpy(wchar_t * dst,
    const wchar_t * src, size_t n);
```

引数 この関数の引数を以下に示します。

dst	wchar_t *	コピー先の文字列
src	wchar_t *	コピー元の文字列
n	size_t	コピーする文字数

注意点 拡張文字用に strncpy 関数と同様の処理を行います。

戻り値 コピー先の文字列へのポインタを返します。

参照 [「strncpy」\(p292 \)](#)、[「wcscpy」\(p368 \)](#)

wcspbrk

拡張文字配列において拡張文字セットが最初に現われる位置を探します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
wchar_t * wcspbrk(const wchar_t * str,
    const wchar_t * set);
```

引数 この関数の引数を以下に示します。

str	wchar_t*	検索される文字列
set	wchar_t *	検索する文字のセット

注意点 拡張文字用に strpbrk 関数と同様の処理を行います。

戻り値 `set` 文字列にあるいずれかの文字と一致した、`str` 文字列の最初の位置へのポインタを返します。一致しない場合、ヌルポインタ (NULL) を返します。

参照 [「strpbrk」\(p294 \)](#)

wcsspn

拡張文字配列において、指定された文字セットの内容と一致した文字の数を求めます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
size_t wcsspn(const wchar_t * str,
               const wchar_t * set);
```

引数 この関数の引数を以下に示します。

<code>str</code>	<code>wchar_t *</code>	検索される文字列
<code>set</code>	<code>const wchar_t *</code>	検索する文字列

注意点 拡張文字用に `strspn` 関数と同様の処理を行います。

戻り値 `set` 文字列にある文字と一致した `str` 文字列内の文字数を返します。

参照 [「strspn」\(p296 \)](#)

wcsrchr

拡張文字配列において指定した拡張文字が最後に見つかった位置を返します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
wchar_t * wcsrchr(const wchar_t * str,
                  wchar_t chr);
```

引数 この関数の引数を以下に示します。

<code>str</code>	<code>const wchar_t *</code>	検索される文字列
<code>chr</code>	<code>wchar_t</code>	検索する文字

注意点 拡張文字用に `strrchr` 関数と同様の処理を行います。

戻り値 文字が見つかった位置へのポインタを返します。失敗した場合、ヌルポインタ (NULL) を返します。

参照 「[strchr](#)」 (p295)

wcsstr

拡張文字配列において指定された文字列を検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`wchar_t * wcsstr(const wchar_t * str,`
`const wchar_t * pat);`

引数 この関数の引数を以下に示します。

str	const wchar_t *	検索される文字列 h
pat	const wchar_t *	検索する文字列

注意点 拡張文字用に `strstr` 関数と同様の処理を行います。

戻り値 str 文字列内で最初に pat 文字列が見つかった位置へのポインタを返します。pat 文字列が見つからない場合、ヌルポインタ (NULL) を返します。

参照 「[strstr](#)」 (p297) 「[wcschr](#)」 (p366)

wcstod

拡張文字を数字へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`double wcstod(wchar_t * str, char ** end);`

引数 この関数の引数を以下に示します。

str	wchar_t *	変換する文字列
end	char **	ヌルではない場合、変換できない最初の文字位置へのポインタ

注意点 拡張文字用に strtod 関数と同様の処理を行います。

戻り値 double 型の浮動小数点値を返します。str を double 値へ変換できない場合、HUGE_VAL (math.h で定義されている) を返し、errno に ERANGE を設定します。

参照「[strtod](#)」(p266)

wcstok

拡張文字配列からトークンを取り出します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`wchar_t * wcstok(wchar_t * str,`
`const wchar_t * set);`

引数 この関数の引数を以下に示します。

str	wchar_t *	変更する文字列
set	wchar_t *	検索する文字のリスト

注意点 拡張文字用に strtok 関数と同様の処理を行います。

戻り値 wcstok() の最初の呼び出しで str 文字列で見つかった最初のトークンの位置へのポインタを返し、見つからない場合はヌルポインタを返します。

さらに str に NULL 文字列を指定して wcstok() を呼び出すと、次のトークンへのポインタを返すか、またはトークンがもうない場合はヌルポインタ (NULL) を返します。

参照「[strtok](#)」(p298)

wcsftime

拡張文字列を時間にフォーマットします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`size_t wcsftime(wchar_t * str,`
`size_t max_size,`
`const wchar_t * format_str,`

```
const struct tm * timeptr);
```

引数 この関数の引数を以下に示します。

str	wchar_t *	コピー先の文字列
max_size	size_t	最大文字列サイズ
format_str	const wchar_t *	フォーマットを定義する文字列
timeptr	const struct tm *	カレンダータイムを含むタイム構造体

注意点 拡張文字用に `strftime` 関数と同様の処理を行います。

戻り値 引数 `str` 内のヌル文字を含む文字数が引数 `max_size` より小さい場合、`wcsftime` 関数は引数 `str` 内の総文字数を返します。大きい場合、`wcsftime` はゼロを返します。

参照 [「strftime」\(p311\)](#)

wcsxfrm

指定した文字数の拡張文字列をコピーします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
size_t wcsxfrm(wchar_t * str1,
               const wchar_t * str2, size_t n);
```

引数 この関数の引数を以下に示します。

str1	wchar_t *	コピー先の文字列
str2	wchar_t *	コピー元の文字列
n	size_t	最大の文字数

注意点 拡張文字用に `strxfrm` 関数と同様の処理を行います。

戻り値 コピーした文字数を返します。

参照 [「strxfrm」\(p300\)](#)

wctime

拡張文字配列を `time_t` 型へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`wchar_t * wctime(const time_t * timer);`

引数 この関数の引数を以下に示します。

timer const time_t * カレンダータイム

注意点 拡張文字用に `ctime` 関数と同様の処理を行います。

戻り値 変換した `time_t` 型を含む拡張文字配列へのポインタを返します。

参照 [「ctime」\(p307\)](#)

wctrans

文字の再マッピングのために `toupper` や `tolower` 属性値を生成します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`wctrans_t wctrans(const char *name);`

引数 この関数の引数を以下に示します。

name const char * toupper または tolower 属性

注意点 拡張文字間のマッピングを表わす値を生成します。

戻り値 `wctrans_t` 型を返します。

参照 [「towctrans」\(p360\)](#)

wmemchr

拡張文字を検索します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`void * wmemchr(const void * src,`
`int val, size_t n);`

引数 この関数の引数を以下に示します。

src	const void *	検索される文字列
val	int	検索する値
n	size_t	最大の検索長

注意点 拡張文字用に `memchr` 関数と同様の処理を行います。

戻り値 検索された拡張文字へのポインタを返します。見つからない場合、ヌルポインタ (NULL) を返します。

参照 [「memchr」\(p276\)](#)、[「wcschr」\(p366\)](#)

wmemcmp

2つのメモリブロックを比較します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`int wmemcmp(const void * src1,`
`const void * src2,`
`size_t n);`

引数 この関数の引数を以下に示します。

src1	const void *	比較文字列その 1
src2	const void *	比較文字列その 2
n	size_t	最大の比較長

注意点 拡張文字用に `memcmp` 関数と同様の処理を行います。

戻り値 文字列 `src1` と `src2` の全 `n` 文字が一致する場合、`wmemcmp` 関数はゼロを返します。`src1` に最初に見つかった一致しない文字が `src2` のそれよりも小さい場合は負の値を返し、大きい場合は正の値を返します。

参照 [「memcmp」\(p278\)](#)、[「wcscmp」\(p366\)](#)

wmemcpy

連続するメモリブロックをコピーします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
void * (wmemcpy)(void * dst,
    const void * src, size_t n);
```

引数 この関数の引数を以下に示します。

dst	void *	コピー先の文字列
src	const void *	コピー元の文字列
n	size_t	最大のコピー長

注意点 拡張文字用に memcpy 関数と同様の処理を行います。

戻り値 コピー先の文字列へのポインタを返します。

参照 「[memcpy](#)」(p279)

wmemmove

オーバーラップする連続メモリブロックをコピーします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
void * (wmemmove)(void * dst,
    const void * src,
    size_t n);
```

引数 この関数の引数を以下に示します。

dst	void *	コピー先の文字列
src	const void *	コピー元の文字列
n	size_t	最大のコピー長

注意点 拡張文字用に memmove 関数と同様の処理を行います。

戻り値 コピー先の文字列へのポインタを返します。

参照 「[memmove](#)」(p279)、 「[wcscpy](#)」(p368)

wmemset

メモリブロックの内容をクリアします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
void * wmemset(void * dst, int val, size_t n);
```

引数 この関数の引数を以下に示します。

dst	void *	コピー先の文字列
val	int	設定する値
n	size_t	最大の長さ

注意点 拡張文字用に `memset` 関数と同様の処理を行います。

戻り値 コピー先の文字列へのポインタを返します。

参照 [「memset」\(p280\)](#)

wprintf

標準出力へフォーマットされたテキストを出力します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wchar.h>
int wprintf(const wchar_t * format, ...);
```

引数 この関数の引数を以下に示します。

format	wchar_t*	フォーマットを定義する文字列
....		可変個の引数

注意点 拡張文字用に `printf` 関数と同様の処理を行います。

戻り値 書き出した引数の個数を返します。エラーの場合、負の値を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353\)](#)、[「printf」\(p211\)](#)、[「fwprintf」\(p354\)](#)

wscanf

標準入力から拡張文字にフォーマットされたテキストを読み込みます。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wchar.h>`
`int wscanf(const wchar_t * format, ...);`

引数 この関数の引数を以下に示します。

format	wchar_t*	フォーマットを定義する文字列
....		可変個の引数

注意点 拡張文字用に `scanf` 関数と同様の処理を行います。

戻り値 読み込みに成功した引数の数を返します。変換フォーマットが一致しない場合、または最後の入力に達した場合、WEOF を返します。

参照 [「拡張文字とバイト文字のストリーム方向」\(p353 \)](#)、[「scanf」\(p224 \)](#)、[「fwscanf」\(p356 \)](#)

第 34 章 wctype.h

wctype.h ヘッダファイルには拡張文字型のテストや操作に関するマクロ定義や関数宣言があります。

wctype.h の概要

wctype.h ヘッダファイルは以下の関数を定義しています。

- 「[iswalnum](#)」: 英数字の拡張文字をテストする
- 「[iswalpha](#)」: 英字の拡張文字をテストする
- 「[iswcntrl](#)」: コントロールの拡張文字をテストする
- 「[iswdigit](#)」: 数字の拡張文字をテストする
- 「[iswgraph](#)」: グラフィックの拡張文字をテストする
- 「[iswlower](#)」: 小文字の拡張文字をテストする
- 「[iswprint](#)」: 印刷可能な拡張文字をテストする
- 「[iswpunct](#)」: 句読点の拡張文字をテストする
- 「[iswspace](#)」: 空白文字の拡張文字をテストする
- 「[iswupper](#)」: 大文字の拡張文字をテストする
- 「[iswxdigit](#)」: 16 進数の拡張文字をテストする
- 「[towlower](#)」: 拡張文字を小文字へ変換する
- 「[towupper](#)」: 拡張文字を大文字へ変換する

iswalnum

英数字の拡張文字であるか否かをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wctype.h>
int iswalnum (wchar_t wc);
```

引数 この関数の引数について以下に説明します。

wc wchar_t テストする拡張文字

注意点 拡張文字用に isalnum 関数と同様の処理を行います。

戻り値 英数字 ([a-z], [A-Z], [0-9]) の場合、真を返します。

参照 「[iswalnum](#)」

iswalpha

英字の拡張文字であるか否かをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wctype.h>
int iswalpha (wchar_t wc);
```

引数 この関数の引数について以下に説明します。

wc wchar_t テストする拡張文字

注意点 拡張文字用に isalpha 関数と同様の処理を行います。

戻り値 英字 ([a-z], [A-Z]) の場合、真を返します。

参照 「[iswalpha](#)」

iswcntrl

コントロールの拡張文字であるか否かをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wctype.h>
int iswcntrl (wchar_t wc);
```

引数 この関数の引数について以下に説明します。

wc wchar_t テストする拡張文字

注意点 拡張文字用に iscntrl 関数と同様の処理を行います。

戻り値 削除文字 (0x7F) または通常のコントロール文字 (0x00 から 0x1F) の場合、真を返します。

参照 [「iswcntrl」](#)

iswdigit

数字の拡張文字であるか否かをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wctype.h>
int iswdigit (wchar_t wc);
```

引数 この関数の引数について以下に説明します。

wc wchar_t テストする拡張文字

注意点 拡張文字用に isdigit 関数と同様の処理を行います。

戻り値 数字 ([0-9]) の場合、真を返します。

参照 [「iswdigit」](#)

iswgraph

グラフィックの拡張文字であるか否かをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wctype.h>
int iswgraph (wchar_t wc);
```

引数 この関数の引数について以下に説明します。

wc wchar_t テストする拡張文字

注意点 拡張文字用に isgraph 関数と同様の処理を行います。

戻り値 ! (0x21) から (0x7E) までのいずれかの印刷不可文字の場合、真を返します。

参照 [「iswgraph」](#)

iswlower

小文字の拡張文字であるか否かをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wctype.h>
int iswlower (wchar_t wc);
```

引数 この関数の引数について以下に説明します。

wc wchar_t テストする拡張文字

注意点 拡張文字用に islower 関数と同様の処理を行います。

戻り値 小文字 ([a-z]) の場合、真を返します。

参照 「[iswlower](#)」

iswprint

印刷可能な拡張文字であるか否かをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wctype.h>
int iswprint (wchar_t wc);
```

引数 この関数の引数について以下に説明します。

wc wchar_t テストする拡張文字

注意点 拡張文字用に isprint 関数と同様の処理を行います。

戻り値 スペース (0x20) から (0x7E) までの印刷可能文字の場合、真を返します。

参照 「[iswprint](#)」

iswpunct

句読点の拡張文字であるか否かをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wctype.h>
int iswpunct (wchar_t wc);
```

引数 この関数の引数について以下に説明します。

wc wchar_t テストする拡張文字

注意点 拡張文字用に ispunct 関数と同様の処理を行います。

戻り値 句読点文字の場合、真を返します。句読点文字はコントロール文字でもなく、英数字でもない文字です。

参照 「[iswpunct](#)」

iswspace

空白文字の拡張文字であるか否かをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ

```
#include <wctype.h>
int iswspace (wchar_t wc);
```

引数 この関数の引数について以下に説明します。

wc wchar_t テストする拡張文字

注意点 拡張文字用に isspace 関数と同様の処理を行います。

戻り値 スペース、タブ、リターン、改行、垂直タブ、フォームフィードのいずれかの場合、真を返します。

参照 「[isspace](#)」

iswupper

大文字の拡張文字であるか否かをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wctype.h>`
`int iswupper (wchar_t wc);`

引数 この関数の引数について以下に説明します。

wc wchar_t テストする拡張文字

注意点 拡張文字用に `isupper` 関数と同様の処理を行います。

戻り値 大文字 ([A-Z]) 場合、真を返します。

参照 [「isupper」](#)

iswxdigit

16 進数文字の拡張文字であるか否かをテストします。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wctype.h>`
`int iswxdigit(wchar_t wc);`

引数 この関数の引数について以下に説明します。

wc wchar_t テストする拡張文字

注意点 拡張文字用に `isxdigit` 関数と同様の処理を行います。

戻り値 16 進数文字 ([0-9], [A-F], [a-f]) のいずれかの場合、真を返します。

参照 [「isxdigit」](#)

towlower

拡張文字を小文字へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wctype.h>`
`wchar_t tolower (wchar_t wc);`

引数 この関数の引数について以下に説明します。

`wc` `wchar_t` 変換される拡張文字

注意点 拡張文字用に `tolower` 関数と同様の処理を行います。

戻り値 大文字を大文字に対応する小文字へ変換し、小文字はそのまま返します。

参照 [「tolower」](#)、[「toupper」](#)

toupper

拡張文字を大文字へ変換します。

互換性 この関数は以下のターゲットに対して互換性があります。

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

プロトタイプ `#include <wctype.h>`
`wchar_t toupper (wchar_t wc);`

引数 この関数の引数を以下に示します。

`wc` `wchar_t` 変換される拡張文字

注意点 拡張文字用に `toupper` 関数と同様の処理を行います。

戻り値 小文字を小文字に対応する大文字へ変換し、大文字はそのまま返します。

参照 [「toupper」](#)、[「tolower」](#)

索引

記号

- `_path2fss` 125
- `_ttyname` 30
- `_vfwscanf` 361
- `_vswscanf` 361
- `_beginthreadex` 127
- `_chdir` 65
- `_chdrive` 66
- `_CRTStartup` 35
- `_DIITerminate` 34
- `_endthreadex` 128
- `_fcreator` 339
- `_fileno` 66
- `_ftype` 340
- `_get_osfhandle` 66
- `_getcwd` 67
- `_HandleTable` 34
- `_heapmin` 68
- `_IOFBF` 230
- `_IOLBF` 230
- `_IONBF` 230
- `_isatty` 68
- `_makepath` 69
- `_open_osfhandle` 69
- `_RunInit` 35
- `_searchenv` 70
- `_SetupArgs` 35
- `_strdate` 311
- `_strdup` 287
- `_stricmp` 288
- `_strnicmp` 293
- `_strrev` 296
- `_strupr` 301

A

- `abort` 244
- `abs` 245
- `acos` 85
- `acosf` 85
- `acosh` 110

- `acosl` 85
- `alloca` 77
- `alloca` 21
- `alloca.h` 21 ~ 22, 77
- `Altivec` 188, 198, 213, 225
 - `longjmp` 130, 131
- `Altivec Extensions`
 - `fprintf` 188, 213, 225
 - `scanf` 198
- `ANSIC` 18
- `Argc` 33
- `Argv` 33
- `asctime` 305
- `asin` 86
- `asinf` 86
- `asinh` 111
- `asinl` 86
- `assert` 23
- `assert.h` 23 ~ 24
- `atan` 86
- `atan2` 87
- `atan2f` 88
- `atan2l` 88
- `atanf` 87
- `atanh` 111
- `atanl` 87
- `atexit` 246
- `atof` 248
- `atoi` 249
- `atoi` 249
- `atol` 249
- `atol` 250

B

- `bsearch` 250

C

- `calloc` 253
- `ccommand` 25
- `ceil` 88
- `ceilf` 89

ceil 89
CHAR_BIT 71
CHAR_MAX 71
CHAR_MIN 71
chdir 320
cin 147
clearerr 172
clock 306
clock_t 303
CLOCKS_PER_SEC 306
close 322
clrscr 27, 146
Command-line Arguments 25
console.h 25 ~ 31
copysign 111
cos 89
cosf 90
cosh 90
coshf 91
coshl 91
cosl 90
cout 147
creat 53
crtl.h 33
ctime 307
ctype.h 37 ~ 46
cuserid 324
Customizing SIOUX 148
Customizing WinSIOUX 144

D

Date and time 303
DBL_DIG 61
DBL_EPSILON 62
DBL_MANT_DIG 61
DBL_MAX 62
DBL_MAX_10_EXP 62
DBL_MAX_EXP 62
DBL_MIN 62
DBL_MIN_10_EXP 62
DBL_MIN_EXP 61
difftime 308
div 255

div_t 47
div_t structure 255

E

EDOM 50
environ 34
EOF 171
ERANGE 50
erf 112
erfc 112
errno 49
errno.h 49 ~ 50
Error number defintiions 50
execl 326
execle 326
execlp 326
execv 326
execve 326
execvp 326
exit 256
exp 91
exp2 113
expf 92
expl 92
expm1 113

F

F_DUPFD 55
fabs 92
fabsf 93
fabsl 93
fclose 173
fcntl 54
fcntl.h 53 ~ 57
fdim 114
fdopen 175
fdopen 341
feof 176
ferror 177
fflush 179
fgetc 180
fgetpos 181
fgets 183

fgetwc 353
fgetws 354
FILE 171
fileno 342
float.h 61 ~ 62
Floating Point Classification Macros 82
Floating Point Math Facilities 85
Floating point mathematics 81
floor 93
floorf 94
floorl 94
FLT_DIG 61
FLT_EPSILON 62
FLT_MANT_DIG 61
FLT_MAX 62
FLT_MAX_10_EXP 62
FLT_MAX_EXP 62
FLT_MIN 62
FLT_MIN_10_EXP 62
FLT_MIN_EXP 61
FLT_RADIX 61
FLT_ROUNDS 61
fmax 114
fmin 114
fmod 94
fmodf 95
fmodl 95
fopen 184
fpclassify 82, 84
fprintf 187
fputc 191
fputs 193
fputwc 355
fputws 356
fread 194
free 257
freopen 196
frexp 95
frexpf 96
frexpl 97
fscanf 197
fseek 201
fsetpos 204

FSp_fopen 63
FSp_fopen. 63
fstat 158
ftell 204
fwprintf 354
fwrite 205
fwscanf 356

G

gamma 115
getc 206
getch 28
getchar 208
getcwd 327
GetHandle 67
getlogin 328
getpid 328
gets 209
getwc 357
getwchar 358
gmtime 309

H

HUGE_VAL 110
HUGE_VAL 81
hypot 115

I

Input Control String 197
Input Conversion Specifiers 197
InstallConsole 28
INT_MAX 71
INT_MIN 71
Integral limits 71
isalnum 38
isalpha 39
isatty 330
isctrl 40
isdigit 40
isfinite 83
isgraph 41
isgreater 97
isgreaterless 97

isless 98
islessequal 98
islower 41
isnan 83
isprint 42
ispunct 42
isspace 43
isunordered 98
isupper 43
iswalnum 381
iswalpha 382
iswcntrl 382
iswdigit 383
iswgraph 383
iswlower 384
iswprint 384
iswpunct 384
iswspace 385
iswupper 385
iswxdigit 386
isxdigit 44

K

kbhit 29
kill 141

L

labs 259
LC_ALL 74
LC_COLLATE 74
LC_CTYPE 74
LC_MONETARY 74
LC_NUMERIC 74
LC_TIME 74
lconv structure 73
LDBL_DIG 61
LDBL_EPSILON 62
LDBL_MANT_DIG 61
LDBL_MAX 62
LDBL_MAX_10_EXP 62
LDBL_MAX_EXP 62
LDBL_MIN 62
LDBL_MIN_10_EXP 62

LDBL_MIN_EXP 61
ldexp 99
ldexpf 100
ldexpl 100
ldiv 260
ldiv_t 47
ldiv_t structure 260
lgamma 116
limits.h 71 ~ 72
Locale specification 73
locale.h 73 ~ 75
localeconv 74
localtime 310
log 100
log10 101
log10f 102
log10l 102
log1p 116
log2 117
logb 117
logf 101
logl 101
LONG_MAX 71
LONG_MIN 72
longjmp 129
lseek 331

M

Marco Piovaneli 146
math.h 79 ~ 109
mblen 261
mbstowcs 261
mbtowc 262
memchr 276
memcmp 278
memcpy 279
memmove 279
memset 280
mkdir 160
mktime 310
modf 102
modff 103
modfl 103

N

NaN 81
nan 118
nearbyint 118
nextafter 119
NULL 167

O

offsetof 167
open 56
Output Control String 187
Output Conversion Specifiers 187

P

path2fss 125
path2fss.h 125
perror 210
pow 103
powf 104
powl 104
printf 211
Process.h 127
ptrdiff_t 167
putc 217
putchar 218
puts 219
putwc 358
putwchar 359

Q

qsort 263
Quiet 81

R

raise 137
rand 264
RAND_MAX 264
read 332
ReadCharsFromConsole 29
realloc 265
remainder 119
remove 220

RemoveConsole 30
remquo 120
rename 221
rewind 222
rint 120
rinttol 121
rmdir 333
round 121
roundtol 122

S

scalb 122
scanf 224
Scanset 199
SCHAR_MAX 71
SCHAR_MIN 71
SEEK_CUR 202
SEEK_END 202
SEEK_SET 202
send_signal 141
setbuf 228
setjmp 130
setjmp.h 129 ~ 131
setlocale 74
setvbuf 229
SHRT_MAX 71
SHRT_MIN 71
SIG_DFL 135
SIG_ERR 135
SIG_IGN 135
SIGABRT 134, 244
sigaction 139
SIGFPE 134
SIGILL 134
SIGINT 134
signal 135
Signal handling 133
signal.h 133 ~ 137
Signaling 81
sigpending 140
sigprocmask 139
SIGSEGV 134
sigsuspend 140

SIGTERM 134
sin 104
sinf 105
sinh 105
sinhf 106
sinhl 106
sinl 105
SIOUX 18, 146
SIOUX.h 155
SIOUXclrscr 153
SIOUXHandleOneEvent 154
SIOUXSettings structure 148
size_t 168
sleep 333
sprintf 231
sqrt 106
sqrtf 107
sqrtl 107
srand 266
sscanf 232
Standard definitions 167
Standard input/output 170
stat 161
stat.h 157 ~ 161
stdarg.h 163 ~ 165
stddef.h 167 ~ 168
stderr 171
stdin 147, 171
stdio.h 169 ~ 241
stdlib.h 243 ~ 273
stdout 147, 171
strcasecmp 281
strcat 281
strchr 282
strcmp 283
strcoll 284
strcpy 285
strcspn 286
strdup 287
Stream Orientation 171, 353
Streams 170
strerror 288
strftime 311

string.h 275 ~ 300
strlen 289
strncasecmp 290
strncat 290
strncmp 291
strncpy 292
strpbrk 294
strrchr 295
strspn 296
strstr 297
strtod 267
strtok 298
strtol 268
strtoul 268
strtol 269
struct vregs 141
strxfrm 300
swprintf 359
swscanf 360
system 270

T

tan 108
tanf 108
tanh 109
tanhf 109
tanhl 110
tanl 109
tell 343
time 316
time.h 303 ~ 313
time_t 303
timeval structure 347
Tm Structure Members. 304
tmpfile 233
tmpnam 234
tolower 45
toupper 46
tolower 386
toupper 387
trunc 123
ttyname 334

tzname 305

tzset 317

U

UCHAR_MAX 71

ULONG_MAX 72

uname 349

ungetc 235

unistd.h 319 ~ 337

unix.h 339 ~ 344

unlink 335

USHRT_MAX 71

Using SIOUX and WinSIOUX 143

utime 345

utime.h 345 ~ 348

utimes 347

utsname structure 350

utsname.h 349 ~ 350

V

va_arg 163

va_end 164

va_list 163

va_start 164

Variable arguments 163

vec_calloc 270

vec_free 271

vec_realloc 271

vfprintf 237

vfwprintf 362

vprintf 239

vsprintf 240

vswprintf 363

vwprintf 364

vwscanf 362

W

WASTE 146

watof 365

wchar_t 168

wscat 365

wcschr 366

wcscmp 366

wscoll 367

wscpy 368

wscspn 367

wcsftime 373

wcslen 368

wcsncat 369

wcsncmp 369

wcsncpy 370

wcspbrk 370

wcsrchr 371

wcsspn 371

wcsstr 372

wctod 372

wctok 373

wctombs 272

wcsxfrm 374

wctime 374

wctomb 273

wctrans 375

WinSIOUX 144

WinSIOUXclrscr 145

wmemchr 375

wmemcmp 376

wmemcpy 376

wmemmove 377

wmemset 378

wprintf 378

write 336

WriteCharsToConsole 31

wscanf 379

CodeWarrior

MSL C Reference

Credits

writing lead: Ron Liechty
other writers: Marc Paquette
engineering: Michael Marcotty, MSL Engineers
frontline warriors: Tech Support, MSL Team
translation: Eiko Kambara, Naomi Owashi
proofreader: Chizu Kanbara

CodeWarrior 文書のガイド

CodeWarrior の文書はツールと同様にモジュールのように構成されています。ツール、言語、ライブラリ、ターゲットごとにマニュアルがあります。各 CodeWarrior 製品によって含まれるマニュアルが異なります。この表に記載されていないマニュアルが含まれることもあります。

コアマニュアル	
IDE User Guide	CodeWarrior IDE と CodeWarrior デバッガの使い方
言語 / コンパイラのマニュアル	
C Compilers Reference	C/C++ フロントエンドコンパイラの情報
Error Reference	コンパイラ / リンカのエラーメッセージのリストおよび解説
Assembler Reference	スタンドアローンアセンブラのシンタックス
Command-Line Tools Reference	Mac OS MPW コンパイラのコマンドラインのオプション
Plugin API Manual	CodeWarrior のプラグインコンパイラ / リンカの API
ライブラリのマニュアル	
MSL C Reference	Metrowerks ANSI 標準 C ライブラリの関数のリファレンス
MSL C++ Reference	Metrowerks ANSI 標準 C++ ライブラリの関数のリファレンス
MFC Reference	Win32 用の Microsoft Foundation Classes リファレンス
Win32 SDK Reference	Win32 API の Microsoft リファレンス
The PowerPlant Book	Mac OS 用アプリケーションフレームワークのガイド
PowerPlant Advanced Topics	PowerPlant での Mac OS プログラミングの高度なテクニック
ターゲットマニュアル	
Targeting Java VM	Java 仮想マシンプログラミングでの CodeWarrior の使い方
Targeting Mac OS	Mac OS プログラミングでの CodeWarrior の使い方
Targeting MIPS	MIPS 組み込みプロセッサプログラミングでの CodeWarrior の使い方
Targeting NEC V800/V850 series	NEC V810/830 プロセッサプログラミングでの CodeWarrior の使い方
Targeting Net Yaroze	Net Yaroze プログラミングでの CodeWarrior の使い方
Targeting PlayStation	PlayStation プログラミングでの CodeWarrior の使い方
Targeting PlayStation2	PlayStation2 プログラミングでの CodeWarrior の使い方
Targeting PowerPC	PPC 組み込みプロセッサプログラミングでの CodeWarrior の使い方
Targeting Windows	Windows プログラミングでの CodeWarrior の使い方

印の付いているマニュアルは日本語訳が用意されています。