



# SunOS リファレンスマニュアル 3 : 基本ライブラリ関数

---

Sun Microsystems, Inc.  
4150 Network Circle  
Santa Clara, CA 95054  
U.S.A.

Part No: 817-3840-10  
2003 年 12 月

Copyright 2003 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

本製品およびそれに関連する文書は著作権法により保護されており、その使用、複製、頒布および逆コンパイルを制限するライセンスのもとにおいて頒布されます。サン・マイクロシステムズ株式会社による事前の許可なく、本製品および関連する文書のいかなる部分も、いかなる方法によっても複製することが禁じられます。

本製品の一部は、カリフォルニア大学からライセンスされている Berkeley BSD システムに基づいていることがあります。UNIX は、X/Open Company, Ltd. が独占的にライセンスしている米国ならびに他の国における登録商標です。フォント技術を含む第三者のソフトウェアは、著作権により保護されており、提供者からライセンスを受けているものです。

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions.

本製品に含まれる HG-MinchoL、HG-MinchoL-Sun、HG-PMinchoL-Sun、HG-GothicB、HG-GothicB-Sun、および HG-PGothicB-Sun は、株式会社リコーがリコービマジクス株式会社からライセンス供与されたタイプフェースマスタをもとに作成されたものです。HeiseiMin-W3H は、株式会社リコーが財団法人日本規格協会からライセンス供与されたタイプフェースマスタをもとに作成されたものです。フォントとして無断複製することは禁止されています。

Sun、Sun Microsystems、docs.sun.com、AnswerBook、AnswerBook2、JumpStart、Solaris Web Start、Power Management、Sun ONE Application Server、Solaris Flash、Solaris Live Upgrade は、米国およびその他の国における米国 Sun Microsystems, Inc. (以下、米国 Sun Microsystems 社とします) の商標もしくは登録商標です。

サンのロゴマークおよび Solaris は、米国 Sun Microsystems 社の登録商標です。

すべての SPARC 商標は、米国 SPARC International, Inc. のライセンスを受けて使用している同社の米国およびその他の国における商標または登録商標です。SPARC 商標が付いた製品は、米国 Sun Microsystems 社が開発したアーキテクチャに基づくものです。

OPENLOOK、OpenBoot、JLE は、サン・マイクロシステムズ株式会社の登録商標です。

Wnn は、京都大学、株式会社アステック、オムロン株式会社で共同開発されたソフトウェアです。

Wnn6 は、オムロン株式会社、オムロンソフトウェア株式会社で共同開発されたソフトウェアです。© Copyright OMRON Co., Ltd. 1995-2000. All Rights Reserved. © Copyright OMRON SOFTWARE Co., Ltd. 1995-2002 All Rights Reserved.

「ATOK」は、株式会社ジャストシステムの登録商標です。

「ATOK Server/ATOK12」は、株式会社ジャストシステムの著作物であり、「ATOK Server/ATOK12」にかかる著作権その他の権利は、株式会社ジャストシステムおよび各権利者に帰属します。

本製品に含まれる郵便番号辞書 (7 桁/5 桁) は郵政事業庁が公開したデータを元に制作された物です (一部データの加工を行なっています)。

本製品に含まれるフェイスマーク辞書は、株式会社ビレッジセンターの許諾のもと、同社が発行する『インターネット・パソコン通信フェイスマークガイド '98』に添付のものを使用しています。© 1997 ビレッジセンター

Unicode は、Unicode, Inc. の商標です。

本書で参照されている製品やサービスに関しては、該当する会社または組織に直接お問い合わせください。

OPEN LOOK および Sun Graphical User Interface は、米国 Sun Microsystems 社が自社のユーザおよびライセンス実施権者向けに開発しました。米国 Sun Microsystems 社は、コンピュータ産業用のビジュアルまたはグラフィカル・ユーザインタフェースの概念の研究開発における米国 Xerox 社の先駆者としての成果を認めるものです。米国 Sun Microsystems 社は米国 Xerox 社から Xerox Graphical User Interface の非独占的ライセンスを取得しており、このライセンスは米国 Sun Microsystems 社のライセンス実施権者にも適用されます。

DtComboBox ウィジェットと DtSpinBox ウィジェットのプログラムおよびドキュメントは、Interleaf, Inc. から提供されたものです。(© 1993 Interleaf, Inc.)

本書は、「現状のまま」をベースとして提供され、商品性、特定目的への適合性または第三者の権利の非侵害の黙示の保証を含みそれに限定されず、明示的であるか黙示的であるかを問わない、なんらの保証も行われぬものとします。

本製品が、外国為替および外国貿易管理法 (外為法) に定められる戦略物資等 (貨物または役務) に該当する場合、本製品を輸出または日本国外へ持ち出す際には、サン・マイクロシステムズ株式会社の事前の書面による承諾を得ることのほか、外為法および関連法規に基づく輸出手続き、また場合によっては、米国商務省または米国所轄官庁の許可を得ることが必要です。

原典: *man pages section 3 : Basic Library Functions*

Part No: 817-0692-10

Revision A



031203@7518



# 目次

---

はじめに 9

**SunOS** リファレンスマニュアル 3: 基本ライブラリ関数 13

Intro(3) 14

asctime(3C) 36

bindtextdomain(3C) 41

bind\_textdomain\_codeset(3C) 46

cftime(3C) 51

cset(3C) 56

csetcol(3C) 57

csetlen(3C) 58

csetno(3C) 59

ctype(3C) 60

dcgettext(3C) 63

dcngettext(3C) 68

dgettext(3C) 73

dngettext(3C) 78

euccol(3C) 83

euclen(3C) 84

eucscol(3C) 85

fgetwc(3C) 86

fgetws(3C) 88

fprintf(3C) 89

fputwc(3C) 98

fputws(3C) 100

fscanf(3C) 101

getdate(3C) 108  
gettext(3C) 114  
getwc(3C) 119  
getwchar(3C) 120  
getwidth(3C) 121  
getws(3C) 122  
iconv(3C) 123  
isalnum(3C) 128  
isalpha(3C) 131  
isascii(3C) 134  
iscntrl(3C) 137  
isdigit(3C) 140  
isenglish(3C) 143  
isgraph(3C) 145  
isideogram(3C) 148  
islower(3C) 150  
isnumber(3C) 153  
isphonogram(3C) 155  
isprint(3C) 157  
ispunct(3C) 160  
isspace(3C) 163  
isspecial(3C) 166  
isupper(3C) 168  
iswalnum(3C) 171  
iswalpha(3C) 173  
iswascii(3C) 175  
iswcntrl(3C) 177  
iswctype(3C) 179  
iswdigit(3C) 181  
iswgraph(3C) 183  
iswlower(3C) 185  
iswprint(3C) 187  
iswpunct(3C) 189  
iswspace(3C) 191  
iswupper(3C) 193  
iswxdigit(3C) 195  
isxdigit(3C) 197  
mblen(3C) 200

mbstowcs(3C)	201
mbtowc(3C)	202
ngettext(3C)	203
printf(3C)	208
putwc(3C)	217
putwchar(3C)	219
putws(3C)	221
scanf(3C)	222
setlocale(3C)	229
snprintf(3C)	232
sprintf(3C)	241
sscanf(3C)	250
strftime(3C)	257
textdomain(3C)	262
toascii(3C)	267
_tolower(3C)	268
tolower(3C)	269
_toupper(3C)	270
toupper(3C)	271
tolower(3C)	272
toupper(3C)	273
ungetwc(3C)	274
vfscanf(3C)	275
vscanf(3C)	282
vsscanf(3C)	289
watof(3C)	296
watoi(3C)	298
watol(3C)	300
watoll(3C)	302
wscat(3C)	304
wcschr(3C)	309
wscmp(3C)	314
wscoll(3C)	319
wscpy(3C)	320
wscspn(3C)	325
wcsetno(3C)	330
wcsftime(3C)	331
wcslen(3C)	332

wcsncat(3C) 337  
wcsncmp(3C) 342  
wcsncpy(3C) 347  
wmspbrk(3C) 352  
wscrchr(3C) 357  
wcsspn(3C) 362  
wcstod(3C) 367  
wcstok(3C) 369  
wcstol(3C) 374  
wcstombs(3C) 376  
wcstoul(3C) 377  
wcstring(3C) 379  
wswcs(3C) 384  
wswidth(3C) 389  
wscxfrm(3C) 390  
wtomb(3C) 392  
wctype(3C) 393  
wcwidth(3C) 394  
windex(3C) 395  
wrindex(3C) 400  
wscasecmp(3C) 405  
wscat(3C) 406  
wschr(3C) 411  
wscmp(3C) 416  
wscol(3C) 421  
wscoll(3C) 422  
wscopy(3C) 423  
wscspn(3C) 428  
wsdup(3C) 433  
wslen(3C) 434  
wsncasecmp(3C) 439  
wsncat(3C) 440  
wsncmp(3C) 445  
wsncpy(3C) 450  
wspbrk(3C) 455  
wsprintf(3C) 460  
wsrchr(3C) 461  
wsscanf(3C) 466

wsspn(3C)	467
wstod(3C)	472
wstok(3C)	474
wstol(3C)	479
wstring(3C)	481
wsxfrm(3C)	482





# はじめに

---

## 概要

SunOS リファレンスマニュアルは、初めて SunOS を使用するユーザーやすでにある程度の知識を持っているユーザーのどちらでも対応できるように解説されています。このマニュアルを構成するマニュアルページは一般に参照マニュアルとして作られており、チュートリアルな要素は含んでいません。それぞれのコマンドを実行すると、どのような結果が得られるかについて、詳しく説明されています。なお、各マニュアルページの内容はオンラインでも参照することができます。

このマニュアルは、マニュアルページの内容によっていくつかのセクションに分かれています。各セクションについて以下に簡単に説明します。

- セクション 1 は、オペレーティングシステムで使えるコマンドを説明します。
- セクション 1M は、システム保守や管理用として主に使われるコマンドを説明します。
- セクション 2 は、すべてのシステムコールについて説明します。ほとんどのシステムコールに 1 つまたは複数のエラーがあります。エラーの場合、通常ありえない戻り値が返されます。
- セクション 3 は、さまざまなライブラリ中の関数について説明します。ただし、UNIX システムプリミティブを直接呼び出す関数については、セクション 2 で説明しています。
- セクション 4 は、各種ファイルの形式について説明します。また、ファイル形式を宣言する C 構造体を適用できる場合には随時説明しています。
- セクション 5 は、文字セットテーブルなど他のセクションには該当しないものについて説明します。
- セクション 7 は、特殊なハードウェア周辺装置またはデバイスドライバに関するさまざまな特殊ファイルについて説明します。STREAMS ソフトウェアドライバ、モジュール、またはシステムコールの STREAMS 汎用セットについても説明します。

- セクション9は、カーネル環境でデバイスドライバを記述するのに必要な参照情報を提供します。ここでは、デバイスドライバインタフェース (DDI) とドライバ/カーネルインタフェース (DKI) という2つのデバイスドライバインタフェース仕様について説明します。
- セクション9Fは、デバイスドライバが使用できるカーネル関数について説明します。

以下に、このマニュアルの項目を表記されている順に説明します。ほとんどのマニュアルページが下記の項目からなる共通の書式で書かれていますが、必要でない項目については省略されています。たとえば、記述すべきバグがコマンドにない場合などは、「使用上の留意点」という項目はありません。各マニュアルページの詳細は各セクションの intro を、マニュアルページの一般的な情報については man(1) を参照してください。

名前                    コマンドや関数の名称と概略が示されています。

形式                    コマンドや関数の構文が示されています。標準パスにコマンドやファイルが存在しない場合は、フルパス名が示されます。字体は、コマンド、オプションなどの定数にはボールド体 (bold) を、引数、パラメータ、置換文字などの変数にはイタリック体 (Italic) または <日本語訳> を使用しています。オプションと引数の順番は、アルファベット順です。特別な指定が必要な場合を除いて、1文字の引数、引数のついたオプションの順に書かれています。

以下の文字がそれぞれの項目で使われています。

- [ ]                    このかっこに囲まれたオプションや引数は省略できます。このかっこが付いていない場合には、引数を必ず指定する必要があります。
- ...                    省略符号。前の引数に変数を付けたり、引数を複数指定したりできることを意味します (例: 'filename...')。
- |                      区切り文字 (セパレータ)。この文字で分割されている引数のうち1つだけを指定できます。
- { }                    この大かっこに囲まれた複数のオプションや引数は省略できます。かっこ内を1組として扱います。

プロトコル              この項が使われているのは、プロトコルが記述されているファイルを示すサブセクション 3R だけです。パス名は常にボールド体 (bold) で示されています。

機能説明                コマンドの機能とその動作について説明します。実行時の詳細を説明していますが、オプションの説明や使用例はここでは示されていません。対話形式のコマンド、サブコマンド、リクエスト、マクロ、関数などに関しては「使用法」で説明します。

IOCTL                  セクション7だけに使用される項です。ioctl(2) システムコールへのパラメータは ioctl と呼ばれ、適切なパラメータを持つデバイスクラスのマニュアルページだけに記載されています。特定の

	<p>デバイスに関する <code>ioctl</code> は、(そのデバイスのマニュアルページに) アルファベット順に記述されています。デバイスの特定のクラスに関する <code>ioctl</code> は、<code>mtio(7I)</code> のように <code>io</code> で終わる名前が付いているデバイスクラスのマニュアルページに記載されています。</p>
オプション	<p>各オプションがどのように実行されるかを説明しています。「形式」で示されている順に記述されています。オプションの引数はこの項目で説明され、必要な場合はデフォルト値を示します。</p>
オペランド	<p>コマンドのオペランドを一覧表示し、各オペランドがコマンドの動作にどのように影響を及ぼすかを説明しています。</p>
出力	<p>コマンドによって生成される出力 (標準出力、標準エラー、または出力ファイル) を説明しています。</p>
戻り値	<p>値を返す関数の場合、その値を示し、値が返される時の条件を説明しています。関数が 0 や -1 のような一定の値だけを返す場合は、値と説明の形で示され、その他の場合は各関数の戻り値について簡単に説明しています。void として宣言された関数はこの項では扱いません。</p>
エラー	<p>失敗の場合、ほとんどの関数はその理由を示すエラーコードを <code>errno</code> 変数の中に設定します。この項ではエラーコードをアルファベット順に記述し、各エラーの原因となる条件について説明します。同じエラーの原因となる条件が複数ある場合は、エラーコードの下にそれぞれの条件を別々のパラグラフで説明しています。</p>
使用法	<p>この項では、使用する際の手がかりとなる説明が示されています。特定の決まりや機能、詳しい説明の必要なコマンドなどが示されています。組み込み機能については、以下の小項目で説明しています。</p>
	<p>コマンド 修飾子 変数 式 入力文法</p>
使用例	<p>コマンドや関数の使用例または使用方法を説明しています。できるだけ実際に入力するコマンド行とスクリーンに表示される内容を例にしています。例の中には必ず <code>example%</code> のプロンプトが出てきます。スーパーユーザーの場合は <code>example#</code> のプロンプトになります。例では、その説明、変数置換の方法、戻り値が示され、それらのほとんどが「形式」、「機能説明」、「オプション」、「使用法」の項からの実例となっています。</p>
環境	<p>コマンドや関数が影響を与える環境変数を記述し、その影響について簡単に説明しています。</p>

終了ステータス	コマンドが呼び出しプログラムまたはシェルに返す値と、その状態を説明しています。通常、正常終了には0が返され、0以外の値はそれぞれのエラー状態を示します。
ファイル	マニュアルページが参照するファイル、関連ファイル、およびコマンドが作成または必要とするファイルを示し、各ファイルについて簡単に説明しています。
属性	属性タイプとその対応する値を定義することにより、コマンド、ユーティリティ、およびデバイスドライバの特性を一覧していません。詳細は <code>attributes(5)</code> を参照してください。
関連項目	関連するマニュアルページ、当社のマニュアル、および一般の出版物が示されています。
診断	エラーの発生状況と診断メッセージが示されています。メッセージはボールド体 ( <b>bold</b> ) で、変数はイタリック体 ( <i>Italic</i> ) または <日本語訳> で示されており、Cロケール時の表示形式です。
警告	作業に支障を与えるような現象について説明しています。診断メッセージではありません。
注意事項	それぞれの項に該当しない追加情報が示されています。マニュアルページの内容とは直接関係のない事柄も参照用に扱っています。ここでは重要な情報については説明していません。
使用上の留意点	すでに発見されているバグについて説明しています。可能な場合は対処法も示しています。

## SunOS リファレンスマニュアル 3: 基本ライブラリ関数

---

### Intro(3)

名前	Intro, intro – 関数とライブラリの紹介
機能説明	<p>このセクションでは、さまざまな Solaris ライブラリにある関数のうち、直接 UNIX システムのプリミティブを呼び出す関数 (このマニュアルページのセクション 2 で説明) 以外の関数について説明します。関数の宣言は、各ページで説明されている <code>#include</code> ファイルから取得できます。このセクションの説明はライブラリ別になっています。各ライブラリは、セクション番号の後ろに入っているライブラリ名 (ライブラリの省略名) で識別できます。関連のあるライブラリのコレクションは、下記に示すように大きく 5 つのボリュームに分かれており、6 番目のボリュームは、この 5 つのボリュームで説明されている関数、マクロ、外部変数が使用する共有ライブラリとヘッダーの内容説明になっています。ここでは、まず、この 6 番目のボリュームから説明します。</p>
ライブラリインタフェースおよびヘッダー	<p>このボリュームでは、他の 5 つのボリュームで説明されている関数、マクロ、外部変数が使用する共有ライブラリとヘッダーの内容について説明します。</p> <p>(3LIB) このセクションで説明するライブラリはどれも、共有オブジェクトとして実装されています。</p> <p>共有オブジェクトの説明には、共有オブジェクトの共有インタフェースを定義する大域シンボルの定義が含まれていることがあります (例: <code>SUNW_1.1</code>)。このほかのインタフェースは、共有オブジェクトに含まれている可能性があります (例: <code>SUNW_private.1.1</code>)。共有インタフェースは、アプリケーション開発用に、安定度の高い専用のシンボルセットを提供します。専用インタフェースは内部使用のみであり、随時変更される可能性があります。</p> <p>共有オブジェクトの多くには、32 ビットのシステムのみで下位互換性を提供するアーカイブライブラリが用意されています。このようなライブラリを使用すると、アプリケーションによっては別の Solaris リリースに移行することができなくなる場合があります。このため、Solaris では、このようなライブラリの代わりにダイナミックリンクを使用するコンパイル方式が優先されます。</p> <p>(3LIBUCB) このセクションで説明する SunOS/BSD 互換ライブラリは、共有オブジェクトとして実装されています。上記の (3LIB) を参照してください。</p> <p>(3HEAD) このセクションで説明するヘッダーは、関数、マクロ、外部変数が使用するヘッダーです。ヘッダーには、関数のプロトタイプ、記号定数の定義、共通の構造体、プリプロセッサマクロ、定義済みの型が含まれています。残りの 5 つのボリュームで説明されている各関数は、この関数を使用するためにアプリケーションに含めなければならないヘッダーを指定します。多くの場合、必要なヘッダーは 1 つだけです。このようなヘッダーは、アプリケーション開発システムに配置されます。実行システムに配置する必要はありません。</p>

## 基本ライブラリ関数

このボリュームで説明する関数は、アプリケーション開発の基本となるコア C ライブラリ関数です。

- (3C) セクション 2 の関数とともに標準 C ライブラリ `libc` を構成する関数です。`libc` は、C コンパイルシステムにより自動的にリンクされます。標準 C ライブラリは共有オブジェクト `libc.so`、アーカイブ `libc.a` として実装されています。C プログラムは、デフォルトで、標準 C ライブラリの共有オブジェクトバージョンにリンクされます。アーカイブバージョンにリンクするには、`cc` コマンド行で `-Bstatic` または `-dn` を指定します。このほかの優先指定については、`libc(3LIB)`、`cc(1B)` を参照してください。  
『ANSI C Programmer's Guide』の「C Compilation System」の章にも参考になる情報があります。規格に合致した環境では、動作が異なる関数もあります。こうした動作については、個々のマニュアルページで説明されています。`standards(5)` を参照してください。
- (3DL) ダイナミックリンクライブラリ `libdl` を構成する関数です。このライブラリは共有オブジェクト `libdl.so` として実装されていますが、C コンパイルシステムによって自動的にリンクされません。`cc` コマンド行で `-ldl` を指定すると、このライブラリにリンクできます。`libdl(3LIB)` を参照してください。
- (3MALLOC) `libmalloc`、`libbsdmalloc`、`libmapmalloc`、`libtmmalloc` という複数のメモリ割り当てライブラリを構成する関数です。これらのライブラリはそれぞれ共有オブジェクト `libmalloc.so`、`libbsdmalloc.so`、`libmapmalloc.so`、`libtmmalloc.so` として実装されています。このうち `libtmmalloc` 以外のすべては、アーカイブ `libmalloc.a`、`libbsdmalloc.a`、`libmapmalloc.a` としても実装されています。このライブラリ群は、C コンパイルシステムによって自動的にリンクされません。`libmalloc`、`libbsdmalloc`、`libmapmalloc`、`libtmmalloc` とリンクするには、それぞれ `-lmalloc`、`-lbsdmalloc`、`-lmapmalloc`、`-lmtmmalloc` の各オプションを指定します。`libmalloc(3LIB)`、`libbsdmalloc(3LIB)`、`libmapmalloc(3LIB)`、および `libtmmalloc(3LIB)` を参照してください。
- (3UCB) BSD 関数を含むソース互換ライブラリを構成する関数です。共有オブジェクト `libucb.so`、アーカイブ `libucb.a` として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリは `/usr/ucb` サブディレクトリにあるため、`cc` コマンド行で `-lucb` を指定することによってリンクできます。このライブラリのヘッダーは、`/usr/ucb/include` に入っています。`libucb(3LIBUCB)` を参照してください。

## ネットワークライブラリ関数

このボリュームで説明する関数は、さまざまなネットワークライブラリを構成します。

### Intro(3)

- (3GSS) このライブラリ中の関数は、Generic Security Services API (GSS-API) ライブラリを構成するルーチンです。このライブラリは共有オブジェクト `libgss.so.1` として実装されています。ただし、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、`cc` コマンド行で `-lgss` を指定します。`libgss(3LIB)` を参照してください。
- (3LDAP) 軽量ディレクトリアクセスプロトコルライブラリを構成する関数です。このライブラリは共有オブジェクト `libldap.so` として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、`cc` コマンド行で `-lldap` を指定します。`ldap(3LDAP)` を参照してください。
- (3NSL) ネットワークサービ斯拉イブラリ `libnsl` を構成する関数です。このライブラリは共有オブジェクト `libnsl.so`、アーカイブ `libnsl.a` として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、`cc` コマンド行で `-lnsl` を指定します。`libnsl(3LIB)` を参照してください。
- 基本ネットワーキング関数の多くは、X/Open ネットワーキングインタフェースライブラリにも入っています。`libxnet` インタフェースの詳細については、下記のセクション (`3XNET`) を参照してください。
- (3RAC) リモート非同期呼び出しライブラリ `librac` を構成する関数です。このライブラリは共有オブジェクト `librac.so`、アーカイブ `librac.a` として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、`cc` コマンド行で `-lrac` を指定します。`librac(3LIB)` を参照してください。
- (3RESOLV) リゾルバライブラリ `libresolv` を構成する関数です。このライブラリは共有オブジェクト `libresolv.so`、アーカイブ `libresolv.a` として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、`cc` コマンド行で `-lresolv` を指定します。`libresolv(3LIB)` を参照してください。
- (3RPC) 遠隔手続き呼び出しライブラリ `librpcsvc` と `librpcsoc` を構成する関数です。`librpcsoc` は互換性の提供のみを目的としたライブラリなので、これには新しいアプリケーションをリンクしないでください。この2つのライブラリは、それぞれ共有オブジェクト `librpcsvc.so` と `librpcsoc.so` として、また、アーカイブ `librpcsvc.a` として実装されています (`librt(3LIB)` 参照)。どちらのライブラリも、C コンパイルシステムによって自動的にリンクされません。これらにリンクするには、`cc` コマンド行で `-lrpcsvc` か `-lrpcsoc` を指定します。`librpcsvc(3LIB)` および `librpcsoc(3LIBUCB)` を参照してください。



- (3SLP) SLP (サービスロケーションプロトコル, Service Location Protocol) ライブラリ `libslp` を構成する関数です。このライブラリは、共有オブジェクト `libslp.so.1` として実装されていますが、C コンパイルシステムによって自動的にリンクされません。 `libslp(3LIB)` を参照してください。
- (3SOCKET) ソケットライブラリ `libsocket` を構成する関数です。このライブラリは、共有オブジェクト `libsocket.so`、アーカイブ `libsocket.a` として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で `-lsocket` を指定します。 `libsocket(3LIB)` を参照してください。
- (3XFN) X/Open フェデレーテッドネーミングライブラリ `libxfn` を構成する関数です。このライブラリは共有オブジェクト `libxfn.so` として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で `-lxfn` を指定します。 `libxfn(3LIB)`、`xfn(3XFN)`、`fns(5)`、`standards(5)` を参照してください。
- (3XNET) X/Open CAE 仕様『Networking Services』(Issue 4、1994年9月)に準拠した X/Open ネットワーキングインタフェースを構成する関数です。このライブラリは共有オブジェクト `libxnet.so` として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で `-lxnet` を指定します。コンパイル情報については、 `libxnet(3LIB)` および `standards(5)` を参照してください。

どのような状況でも、XTI API や TLI API ではなく、ソケット API を優先的に使用することをお勧めします。ただし、ほかの XPGV4v2 (`standards(5)` を参照) システムに移植する必要がある場合は、アプリケーションで `libxnet` インタフェースを使用する必要があります。移植の必要がない場合は、`libxnet` のインタフェースではなく、`libsocket` や `libnsl` のソケットインタフェースを使用する方法をお勧めします。XTI API、TLI API の中では、`libnsl` で使用できる TLI インタフェースよりも `libxnet` で使用できる XTI インタフェースのほうをお勧めします。

#### Curses ライブラリ関数

このボリュームで説明する関数は、グラフィックスと文字が表示された画面の更新機能を提供するライブラリを構成します。

- (3CURSES) 次のライブラリを構成する関数です。
- |                        |  |
|------------------------|--|
| <code>libcurses</code> | <code>curses</code> ライブラリ <code>libcurses</code> を構成する関数です。このライブラリは共有オブジェクト <code>libcurses.so</code> 、アーカイブ <code>libcurses.a</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で <code>-lcurses</code> を指定します。 <code>libcurses(3LIB)</code> を参照してください。 |
|------------------------|--|

Intro(3)

libform	forms ライブラリ libform を構成する関数です。このライブラリは共有オブジェクト libform.so、アーカイブ libforms.a として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で -lform を指定します。libform(3LIB) を参照してください。
libmenu	menus ライブラリ libmenu を構成する関数です。このライブラリは共有オブジェクト libmenu.so、アーカイブ libmenu.a として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で -lmenu を指定します。libmenu(3LIB) を参照してください。
libpanel	panels ライブラリ libpanel を構成する関数です。このライブラリは共有オブジェクト libpanel.so として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で -lpanel を指定します。libpanel(3LIB) を参照してください。
(3PLOT)	グラフィックスライブラリ libplot を構成する関数です。このライブラリは共有オブジェクト libplot.so、アーカイブ libplot.a として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で -lplot を指定します。libplot(3LIB) を参照してください。
(3XCURSES)	/usr/xpg4/lib/libcurses.so.1 にある X/Open Curses ライブラリを構成する関数です。このライブラリは、端末画面での入力と出力、その変更を使用する国際化された関数とマクロのセットを提供します。このライブラリには、ウィンドウの作成、テキストの強調表示、画面への書き込み、ユーザーの入力の読み取り、カーソルの移動などを実行するための関数が含まれています。X/Open Curses は、画面更新動作を最適化するように設計されています。X/Open Curses ライブラリは、X/Open Extended Curses 仕様 (Issue 4) に全面的に準拠しています。
スレッドおよびリアルタイムライブラリ関数	このボリュームで説明する関数は、スレッドおよびリアルタイムライブラリを構成します。
(3AIO)	非同期入出力ライブラリ liaio を構成する関数です。このライブラリは共有オブジェクト libaio.so として実装されていますが、C コンパイルシステムによって自動的にリンクされませ

- ん。このライブラリにリンクするには、cc コマンド行で `-laio` を指定します。libaio(3LIB) を参照してください。
- (3DOOR) doors ライブラリ libdoor を構成する関数です。このライブラリは共有オブジェクト libdoor.so として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で `-ldoor` を指定します。
- (3RT) POSIX.4 リアルタイムライブラリ librt を構成する関数です。このライブラリは共有オブジェクト librt.so として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で `-lrt` を指定します。このライブラリは、以前は libposix4 という名前でした。下位互換のため、この libposix4 も維持されていますが、こちらの使用はできるだけ避けてください。librt(3LIB) を参照してください。
- (3SCHED) LWP スケジューリングライブラリ libsched を構成する関数です。このライブラリは共有オブジェクト libsched.so として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で `-lsched` を指定します。
- (3THR) スレッドライブラリ libpthread、libthread、libthread\_db を構成する関数です。libpthread ライブラリと libthread ライブラリは、マルチスレッドアプリケーションの構築に使用します。libpthread は POSIX (standards(5) を参照) スレッドインタフェースを実装していますが、libthread が実装しているのは Solaris スレッドインタフェースです。libthread\_db ライブラリは、マルチスレッドアプリケーションのデバッグの構築に役立ちます。
- POSIX スレッドと Solaris スレッドの実装には互換性があります。このため、双方のスレッドを同じアプリケーション内で使用することができます。ただし、POSIX 準拠の別の環境への移植性が保証されているのは POSIX スレッドの方だけです。
- POSIX スレッドと Solaris スレッドを同じアプリケーション内で使用しているとき、同じ名前でありながら意味解釈の異なる呼び出しがあると、Solaris スレッドではなく POSIX スレッドを指すものと解釈されます。たとえば、`fork()` の呼び出しは、POSIX スレッドライブラリにリンクしたプログラム内の `fork1()` を意味することになります。このプログラムが POSIX スレッドと同時に Solaris スレッド `-lthread` にもリンクしていたとしても、呼び出しが示しているのは POSIX スレッドの方になります。

## Intro(3)

libpthread、libthread、libthread\_db の各ライブラリは、それぞれ共有オブジェクト libpthread.so、libthread.so、libthread\_db.so として実装されていますが、アーカイブライブラリとしても実装されているのは libthread\_db のみです。このアーカイブの名前は libthread\_db.a です。これらのライブラリはいずれも、C コンパイルシステムによって自動的に実装されてはいません。このライブラリ群にリンクするには、cc コマンド行で、-lpthread、-lthread、-lthread\_db のいずれかを指定します。libpthread(3LIB)、libthread(3LIB)、libthread\_db(3LIB) を参照してください。

次の関数は、POSIX で使用するオプションであり、現在の Solaris リリースではサポートされていません。

```
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
                                  int protocol);

int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
                                  *attr,
                                  int *protocol);

int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
                                      int prioceiling);

int pthread_mutexattr_getprioceiling
(const pthread_mutexattr_t *attr,
 int *prioceiling);
```

### 拡張ライブラリ関数

このボリュームで説明する関数は、特化されたさまざまなライブラリを構成します。構成されるライブラリは、次に示す限りではありません。

- (3BSM) 基本セキュリティライブラリ libbsm を構成する関数です。このライブラリは共有オブジェクト libbsm.so、アーカイブ libbsm.a として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で -lbsm を指定します。libbsm(3LIB) を参照してください。
- (3CFGADM) 構成管理ライブラリ libcfgadm を構成する関数です。このライブラリは共有オブジェクト libcfgadm.so として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で -lcfgadm を指定します。libcfgadm(3LIB) を参照してください。
- (3CPC) CPU パフォーマンスカウンタライブラリ libcpc およびプロセスコンテキストライブラリ libpctx を構成する関数です。これらのライブラリはそれぞれ共有オブジェクト libcpc.so および libpctx.so として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンク

	<p>するには、cc コマンド行で <code>-lcpc</code> を指定します。libcpc(3LIB) および libpctx(3LIB)を参照してください。</p>
(3DEVID)	<p>デバイス ID ライブラリ libdevid を構成する関数です。このライブラリは共有オブジェクト libdevid.so として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で <code>-ldevid</code> を指定します。libdevid(3LIB) を参照してください。</p>
(3DEVINFO)	<p>デバイス情報ライブラリ libbsm を構成する関数です。このライブラリは共有オブジェクト libdevinfo.so、アーカイブ libdevinfo.a として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で <code>-ldevinfo</code> を指定します。libdevinfo(3LIB) を参照してください。</p>
(3DMI)	<p>DMI ライブラリ libdmi、libdmici、libdmimi を構成する関数です。これらのライブラリはそれぞれ、共有オブジェクト libdmi.so、libdmici.so、libdmimi.so として実装されていますが、C コンパイルシステムによって自動的にリンクされません。これらにリンクするには、cc コマンド行で <code>-ldmi</code>、<code>-ldmici</code>、<code>-ldmimi</code> のいずれかを指定します。libdmi(3LIB)、libdmici(3LIB)、libdmimi(3LIB) を参照してください。</p>
(3ELF)	<p>ELF (Extensible Linking Format) アクセスライブラリ libelf を構成する関数です。このライブラリは elf ファイルの作成と分析用のインタフェースを提供します。elf ファイルには、実行ファイル、オブジェクト、共有オブジェクトがあります。libelf は共有オブジェクト libelf.so、アーカイブ libelf.a として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で <code>-lelf</code> を指定します。libelf(3LIB) を参照してください。</p>
(3EXACCT)	<p>拡張アカウンティングアクセスライブラリ libexacct およびプロジェクトデータベースアクセスライブラリ libproject を構成する関数です。これらのライブラリはそれぞれ、共有オブジェクト libexacct.so および libproject.so として実装されていますが、C コンパイルシステムによって自動的にリンクされません。これらのライブラリにリンクするには、cc コマンド行で <code>-lexacct</code> または <code>-lproject</code> を指定します。libexacct(3LIB) および libproject(3LIB) を参照してください。</p>
(3GEN)	<p>文字列のパターンマッチングやパス名の操作を行うライブラリ libgen を構成する関数です。このライブラリは共有オブジェクト libgen.so、アーカイブ libgen.a として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で <code>-lgen</code> を指定します。libgen(3LIB) を参照してください。</p>

## Intro(3)

(3KSTAT)	カーネル統計情報ライブラリを構成する関数です。このライブラリは共有オブジェクト <code>libkstat.so</code> 、アーカイブ <code>libkstat.a</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、 <code>cc</code> コマンド行で <code>-lkstat</code> を指定します。 <code>libkstat(3LIB)</code> を参照してください。
(3KVM)	カーネルの仮想メモリーライブラリへのアクセスを許可する関数です。このライブラリは共有オブジェクト <code>libkvm.so</code> 、アーカイブ <code>libkvm.a</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、 <code>cc</code> コマンド行で <code>-lkvm</code> を指定します。 <code>libkvm(3LIB)</code> を参照してください。
(3LAYOUT)	レイアウトサービスライブラリを構成する関数です。このライブラリは共有オブジェクト <code>liblayout.so</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、 <code>cc</code> コマンド行で <code>-llayout</code> を指定します。 <code>liblayout(3LIB)</code> を参照してください。
(3LGRP)	ローカリティグループライブラリを構成する関数です。このライブラリは共有オブジェクト <code>liblgrp.so</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、 <code>cc</code> コマンド行で <code>-llgrp</code> を指定します。 <code>liblgrp(3LIB)</code> を参照してください。
(3M)	数学ライブラリ <code>libm</code> を構成する関数です。このライブラリは共有オブジェクト <code>libm.so</code> 、アーカイブ <code>libm.a</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、 <code>cc</code> コマンド行で <code>-lm</code> を指定します。
(3MAIL)	ユーザーのメールボックス管理ライブラリ <code>libmail</code> を構成する関数です。このライブラリは共有オブジェクト <code>libmail.so</code> 、アーカイブ <code>libmail.a</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、 <code>cc</code> コマンド行で <code>-lmail</code> を指定します。
(3MP)	整数型数学ライブラリ <code>libmp</code> を構成する関数です。このライブラリは共有オブジェクト <code>libmp.so</code> 、アーカイブ <code>libmp.a</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、 <code>cc</code> コマンド行で <code>-lmp</code> を指定します。 <code>libmp(3LIB)</code> を参照してください。
(3NVP AIR)	名前と値の組み合わせのライブラリ <code>libnvpair</code> を構成する関数です。このライブラリは共有オブジェクト <code>libnvpair.so</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。

	<p>リンクされません。このライブラリにリンクするには、<code>cc</code> コマンド行で <code>-lnvpair</code> を指定します。<code>libnvpair(3LIB)</code> を参照してください。</p>
(3PAM)	<p>PAM (Pluggable Authentication Module) ライブラリ <code>libpam</code> を構成する関数です。このライブラリは共有オブジェクト <code>libpam.so</code>、アーカイブ <code>libpam.a</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、<code>cc</code> コマンド行で <code>-lpam</code> を指定します。<code>libpam(3LIB)</code> を参照してください。</p>
(3PICL)	<p>PICL ライブラリ <code>libpicl</code> を構成する関数です。このライブラリは共有オブジェクト <code>libpicl.so</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、<code>cc</code> コマンド行で <code>-lpicl</code> を指定します。<code>libpicl(3LIB)</code>、<code>libpicl(3LIB)</code> および <code>libpicl(3PICL)</code> を参照してください。</p>
(3PICLTREE)	<p>PICL プラグインライブラリ <code>libpicltree</code> を構成する関数です。このライブラリは共有オブジェクト <code>libpicltree.so</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、<code>cc</code> コマンド行で <code>-lpicltree</code> を指定します。<code>libpicltree(3LIB)</code>、<code>libpicltree(3LIB)</code> および <code>libpicltree(3PICLTREE)</code> を参照してください。</p>
(3RSM)	<p>リモート共有メモリー <code>librsm</code> を構成する関数です。このライブラリは共有オブジェクト <code>librsm.so</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、<code>cc</code> コマンド行で <code>-lrsm</code> を指定します。<code>librsm(3LIB)</code> を参照してください。</p>
(3SEC)	<p>ファイルアクセス制御ライブラリ <code>libsec</code> を構成する関数です。このライブラリは共有オブジェクト <code>libsec.so</code>、アーカイブ <code>libsec.a</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、<code>cc</code> コマンド行で <code>-lsec</code> を指定します。<code>libsec(3LIB)</code> を参照してください。</p>
(3SECDB)	<p>セキュリティ属性データベースライブラリ <code>libsecdb</code> を構成する関数です。このライブラリは共有オブジェクト <code>libsecdb.so</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、<code>cc</code> コマンド行で <code>-lsecdb</code> を指定します。<code>libsecdb(3LIB)</code> を参照してください。</p>
(3SNMP)	<p>SNMP ライブラリ <code>libdssagent</code> と <code>libdssasmp</code> を構成する関数です。この2つのライブラリは、それぞれ共有オブジェクト <code>libssagent.so</code>、<code>libssasmp.so</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされませ</p>

## Intro(3)

	<p>ん。これらにリンクするには、cc コマンド行で <code>-lssagent</code> または <code>-lssasmp</code> を指定します。libssagent(3LIB) を参照してください。</p>
(3SYSEVENT)	<p>システムイベントライブラリ <code>libsysevent</code> を構成する関数です。このライブラリは共有オブジェクト <code>libsysevent.so</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で <code>-lsysevent</code> を指定します。libsysevent(3LIB) を参照してください。</p>
(3TNF)	<p>TNF ライブラリ <code>libtnf</code>、<code>libtnfctl</code>、<code>libtnfprobe</code> を構成する関数です。この3つのライブラリは、それぞれ共有オブジェクト <code>libtnf.so</code>、<code>libtnfctl.so</code>、<code>libtnfprobe.so</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。これらにリンクするには、cc コマンド行で <code>-ltnf</code>、<code>-ltnfctl</code>、<code>-ltnfprobe</code> のいずれかを指定します。libtnfctl(3TNF) および libtnfctl(3LIB) を参照してください。</p>
(3VOLMGT)	<p>ボリューム管理ライブラリ <code>libvolmgt</code> を構成する関数です。このライブラリは共有オブジェクト <code>libvolmgt.so</code>、アーカイブ <code>libvolmgt.a</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で <code>-lvolmgt</code> を指定します。libvolmgt(3LIB) を参照してください。</p>
(3WSREG)	<p>製品インストールレジストリのライブラリ <code>libwsreg</code> を構成する関数です。このライブラリは、共有オブジェクト <code>libwsreg.so</code> として実装されていますが、C コンパイルシステムによって自動的にリンクされません。このライブラリにリンクするには、cc コマンド行で <code>-lwsreg</code> オプションを指定してください。libwsreg(3LIB) を参照してください。</p>
定義	<p>マシンのバイトに適合するビットパターンを、1文字と解釈することができます。ただし、言語によっては、1文字に1バイト以上必要で、複数バイトで表現されるものもあります。</p> <p>NULL 文字とは、値が0の文字のことです。C 言語では、便宜上この文字を <code>\0</code> と表します。文字の並び(シーケンス)のことを文字配列と言います。最後の文字が NULL 文字になっている文字の並びのことを NULL 終了文字配列(文字列)と言います。また、NULL 文字だけを含む文字配列のことを NULL 文字列と言います。0 をポインタにキャストすることによって得られる値は NULL ポインタと呼ばれています。C 言語では、この値は有効なポインタと一致しません。このため、ポインタを返す関数のほとんどは、NULL を返すことによってエラーを示します。NULL マクロは <code>&lt;stdio.h&gt;</code> で定義されています。size_t の型は適切なヘッダーの中に定義されています。</p>
MT レベルのライブラリファイル	<p>MT レベルのライブラリの詳細については <code>attributes(5)</code> を参照してください。</p> <p><code>INCDIR</code> 通常 <code>/usr/include</code></p>



	<code>LIBDIR</code>	通常 <code>/usr/lib</code> (32 ビット) または <code>/usr/lib/sparcv9</code> (64 ビット)
	<code>LIBDIR/libc.so</code>	
	<code>LIBDIR/libc.a</code>	
	<code>LIBDIR/libgen.a</code>	
	<code>LIBDIR/libm.a</code>	
	<code>LIBDIR/libsfm.sa</code>	
	<code>/usr/lib/libc.so.1</code>	
関連項目	<code>ar(1)</code> 、 <code>cc(1B)</code> 、 <code>ld(1)</code> 、 <code>fork(2)</code> 、 <code>intro(3)</code> 、 <code>stdio(3C)</code> 、 <code>attributes(5)</code> 、 <code>standards(5)</code>	
	『リンカーとライブラリ』	
	『プロファイリングツール』	
	『ANSI C Programmer's Guide』	
診断	浮動小数点の値を返す関数の場合、エラー処理はコンパイルのモードによって異なります。 <code>cc</code> に <code>-xt</code> オプションが指定されている場合 (デフォルト) に、指定された引数 が使用する関数が未定義であったり値を表現することができなかつたりすると、これ らの関数は <code>0</code> 、 <code>±HUGE</code> 、 <code>NaN</code> という一定の値を返します。 <code>-xa</code> や <code>-xc</code> のコンパイル モードでは、 <code>±HUGE</code> の代わりに <code>±HUGE_VAL</code> が返されます。 <code>HUGE_VAL</code> と <code>HUGE</code> は <code>math.h</code> の中で定義されています。 <code>HUGE_VAL</code> は値が無限であることを表し、 <code>HUGE</code> は単精度小数点の最大値を表します。	
マルチスレッドア プリケーションに 関する注意事項	マルチスレッドアプリケーションをコンパイルする際は、コマンド行で <code>_POSIX_C_SOURCE</code> フラグ、 <code>_POSIX_PTHREAD_SEMANTICS</code> フラグ、 <code>_REENTRANT</code> フラグのいずれかのフラグを定義する必要があります。これにより、マルチスレッド のアプリケーションだけに適用される関数の定義が特別に有効になります。 <code>POSIX.1003.1c</code> に適合するアプリケーションには、 <code>_POSIX_C_SOURCE</code> フラグに <code>199506L</code> 以上の値を定義してください。	
	<pre>cc [flags] file. . . -D_POSIX_C_SOURCE=199506L -pthread</pre>	
	Solaris の <code>fork()</code> と <code>fork1()</code> 関数を使用した POSIX スレッドは、次のようにコン パイルしてください。	
	<pre>cc [flags] file. . . -D_POSIX_PTHREAD_SEMANTICS -pthread</pre>	
	Solaris スレッドは、次のようにコンパイルしてください。	
	<pre>cc [flags] file... -D_REENTRANT -pthread</pre>	

### Intro(3)

シングルスレッドのアプリケーションを作成する際、上記のフラグを定義してはいけません。定義しないことにより、マルチスレッドをサポートしていない旧リリースの Solaris 上で実行可能なバイナリが生成されます。

アプリケーションを安全に実行するため、メインスレッド以外からは `Unsafe` インタフェースを呼び出さないようにしてください。

MT-Safe インタフェースは、各関数やライブラリのマニュアルページの「属性」の項目に記述されています (`attributes(5)` を参照)。インタフェースが MT-Safe であることがマニュアルページ中に明示されていないときは、そのインタフェースを `Unsafe` とみなしてください。

#### リアルタイムアプリケーション

早期のバインディングを可能にするためには、環境変数 `LD_BIND_NOW` の値を非 NULL 値に設定してください。詳細については『リンカーとライブラリ』の再配置に関する説明を参照してください。

#### 注意事項

ここで説明する関数、外部変数、マクロは、ユーザープログラムの中で再定義しないでください。他の名前を再定義してもこれらのライブラリの動作に影響を与えませんが、同じ名前を再定義すると、インクルードされているヘッダーの宣言と衝突してしまいます。

`INCDIR` 中にあるヘッダーは、このマニュアルページで説明するほとんどの関数のプロトタイプを提供します。関数のプロトタイプとは、引数の型を含む関数宣言のことです。関数のプロトタイプを使用すると、ユーザーのプログラムの中で使用されている関数の使用法が間違っていないか、コンパイラでチェックすることができます。コンパイラの代わりに `lint` プログラムチェッカーを使用することもできます。`lint` は、ヘッダーが `#include` 文でインクルードされていないことを検出すると、この矛盾を報告します。セクション 2、3C、3S の定義は自動的にチェックされます。その他の定義をチェックするには、`lint` に `-1` オプションを指定します。たとえば、`libm` の定義をチェックするには `-lm` を指定します。できるだけ `lint` を使用するようしてください。詳細は、『プロファイリングツール』の `lint` に関する章を参照してください。

「STREAMS」と「ストリーム」の違いには十分気を付けてください。`STREAMS` とは、ネットワークサービスとデータ通信ドライバの開発をサポートするカーネル機構の一種であり、ユーティリティルーチン、カーネル機能、データ構造体で構成されています。一方、ストリームとは、`STREAMS` に関連するバッファリングで使用されるファイルのことです。ストリームは `<stdio.h>` の中で `FILE` 型へのポインタとして定義されています。

各要素の詳細な定義では、実装に固有のシンボリック名を参照しなければならない場合があります。ただし、必ずしもアプリケーションプログラムで使用できるようにする必要はありません。こうしたシンボリック名の多くは、境界の条件やシステムの制限を記述するものです。

このセクションでは、分かりやすくするために、このような実装固有の値をシンボリック名にしています。このようなシンボリック名は必ず中括弧 (`{}`) で囲まれ、ヘッダーを使ってアプリケーションからアクセスできる実装固有の定数のシンボ

リック名と区別されます。中括弧で囲まれているシンボリック名は、特定のシステムの文書中で定義されていることもありますが、ヘッダーを使用することによってアプリケーションプログラムから必ずしもアクセスできるわけではありません。

一般には、移植可能なアプリケーションプログラムコードの中で、このようなシンボリック名が参照されてはなりません。たとえば、あるアプリケーションプログラムで、あるルーチンに提供された引数リストの長さが {ARG\_MAX} より大きいかどうかをテストしてはなりません。

## コマンド一覧

名前	説明
Intro(3)	関数とライブラリの紹介
addnwstr(3curses)	curls_addwstr(3CURSES) を参照
addwch(3curses)	curls_addwch(3CURSES) を参照
addwchnstr(3curses)	curls_addwchstr(3CURSES) を参照
addwchstr(3curses)	curls_addwchstr(3CURSES) を参照
addwstr(3curses)	curls_addwstr(3CURSES) を参照
adjcurspos(3curses)	curls_alecompat(3CURSES) を参照
advance(3gen)	regexpr(3GEN) を参照
ascftime(3c)	strftime(3C) を参照
bindtextdomain(3c)	gettext(3C) を参照
bind_textdomain_codeset(3c)	gettext(3C) を参照
cftime(3c)	strftime(3C) を参照
compile(3gen)	regexpr(3GEN) を参照
config_admin(3CFGADM)	構成の管理
config_ap_id_cmp(3cfgadm)	config_admin(3CFGADM) を参照
config_change_state(3cfgadm)	config_admin(3CFGADM) を参照
config_list(3cfgadm)	config_admin(3CFGADM) を参照
config_list_ext(3cfgadm)	config_admin(3CFGADM) を参照
config_private_func(3cfgadm)	config_admin(3CFGADM) を参照
config_stat(3cfgadm)	config_admin(3CFGADM) を参照
config_strerror(3cfgadm)	config_admin(3CFGADM) を参照
config_test(3cfgadm)	config_admin(3CFGADM) を参照
config_unload_libs(3cfgadm)	config_admin(3CFGADM) を参照
cset(3C)	EUC コードセットに関する情報の取得
csetcol(3c)	cset(3C) を参照

## Intro(3)

<code>csetlen(3c)</code>	<code>cset(3C)</code> を参照
<code>csetno(3c)</code>	<code>cset(3C)</code> を参照
<code>ctype(3C)</code>	文字処理
<code>curses(3CURSES)</code>	CRT 画面操作および最適化用パッケージ
<code>curl_ addwch(3CURSES)</code>	<code>wchar_t</code> 文字を属性とともに <code>curses</code> ウィンドウに追加してカーソルを進める
<code>curl_ addwchstr(3CURSES)</code>	<code>wchar_t</code> 文字列を属性とともに <code>curses</code> ウィンドウに追加
<code>curl_ addwstr(3CURSES)</code>	<code>wchar_t</code> 文字列を <code>curses</code> ウィンドウに追加してカーソルを進める
<code>curl_ alecompat(3CURSES)</code>	カーソルを文字単位に移動するために ALE <code>curses</code> ライブラリに追加された関数
<code>curl_ getwch(3CURSES)</code>	<code>curses</code> 端末キーボードから <code>wchar_t</code> 文字を得る (または戻す)
<code>curl_ getwstr(3CURSES)</code>	<code>curses</code> 端末キーボードから <code>wchar_t</code> 文字列を得る
<code>curl_ inswch(3CURSES)</code>	<code>curses</code> ウィンドウ内のカーソル位置の文字の直前に <code>wchar_t</code> 文字を挿入
<code>curl_ inswstr(3CURSES)</code>	<code>curses</code> ウィンドウ内のカーソル位置の文字の直前に <code>wchar_t</code> 文字列を挿入
<code>curl_ inwch(3CURSES)</code>	<code>curses</code> ウィンドウから <code>wchar_t</code> 文字とその属性を得る
<code>curl_ inwchstr(3CURSES)</code>	<code>curses</code> ウィンドウから <code>wchar_t</code> 文字列とその属性を得る
<code>curl_ inwstr(3CURSES)</code>	<code>curses</code> ウィンドウから <code>wchar_t</code> 文字列を得る
<code>curl_ pad(3CURSES)</code>	<code>curses</code> パッドの作成と表示
<code>dcgettext(3c)</code>	<code>gettext(3C)</code> を参照
<code>dcngettext(3c)</code>	<code>gettext(3C)</code> を参照
<code>dgettext(3c)</code>	<code>gettext(3C)</code> を参照
<code>dnggettext(3c)</code>	<code>gettext(3C)</code> を参照
<code>echowchar(3curses)</code>	<code>curl_ addwch(3CURSES)</code> を参照
<code>euocol(3c)</code>	<code>euclen(3C)</code> を参照
<code>euclen(3C)</code>	EUC 文字のバイト長および表示幅の取得
<code>euocol(3c)</code>	<code>euclen(3C)</code> を参照

<code>fgetwc(3C)</code>	ストリームからワイド文字コードを読み込む
<code>fgetws(3c)</code>	<code>getws(3C)</code> を参照
<code>fprintf(3c)</code>	<code>printf(3C)</code> を参照
<code>fputwc(3C)</code>	ワイド文字コードをストリームへ書き出す
<code>fputws(3C)</code>	ワイド文字列をストリームへ書き出す
<code>fscanf(3c)</code>	<code>scanf(3C)</code> を参照
<code>getdate(3C)</code>	ユーザー形式の日付および時刻の変換
<code>getnwstr(3curses)</code>	<code>curl_getwstr(3CURSES)</code> を参照
<code>gettext(3C)</code>	メッセージ処理関数
<code>getwc(3C)</code>	ストリームからワイド文字を読み込む
<code>getwch(3curses)</code>	<code>curl_getwch(3CURSES)</code> を参照
<code>getwchar(3C)</code>	<code>stdin</code> ストリームからワイド文字を読み込み
<code>getwidth(3C)</code>	コードセット情報の取得
<code>getws(3C)</code>	ストリームからワイド文字列を取得する
<code>getwstr(3curses)</code>	<code>curl_getwstr(3CURSES)</code> を参照
<code>iconv(3C)</code>	コード変換関数
<code>innwstr(3curses)</code>	<code>curl_inwstr(3CURSES)</code> を参照
<code>insnwstr(3curses)</code>	<code>curl_inswstr(3CURSES)</code> を参照
<code>inswch(3curses)</code>	<code>curl_inswch(3CURSES)</code> を参照
<code>inswstr(3curses)</code>	<code>curl_inswstr(3CURSES)</code> を参照
<code>inwch(3curses)</code>	<code>curl_inwch(3CURSES)</code> を参照
<code>inwchnstr(3curses)</code>	<code>curl_inwchstr(3CURSES)</code> を参照
<code>inwchstr(3curses)</code>	<code>curl_inwchstr(3CURSES)</code> を参照
<code>inwstr(3curses)</code>	<code>curl_inwstr(3CURSES)</code> を参照
<code>isalnum(3c)</code>	<code>ctype(3C)</code> を参照
<code>isalpha(3c)</code>	<code>ctype(3C)</code> を参照
<code>isascii(3c)</code>	<code>ctype(3C)</code> を参照
<code>isctrl(3c)</code>	<code>ctype(3C)</code> を参照
<code>isdigit(3c)</code>	<code>ctype(3C)</code> を参照
<code>isencrypt(3GEN)</code>	文字のバッファが暗号化されているかの判別

## Intro(3)

isenglish(3c)	iswalpha(3C) を参照
isgraph(3c)	ctype(3C) を参照
isideogram(3c)	iswalpha(3C) を参照
islower(3c)	ctype(3C) を参照
isnumber(3c)	iswalpha(3C) を参照
isphonogram(3c)	iswalpha(3C) を参照
isprint(3c)	ctype(3C) を参照
ispunct(3c)	ctype(3C) を参照
isspace(3c)	ctype(3C) を参照
isspecial(3c)	iswalpha(3C) を参照
isupper(3c)	ctype(3C) を参照
iswalnum(3c)	iswalpha(3C) を参照
iswalpha(3C)	ワイド文字の文字分類用関数
iswascii(3c)	iswalpha(3C) を参照
iswcntrl(3c)	iswalpha(3C) を参照
iswctype(3C)	指定された文字のクラスの検査
iswdigit(3c)	iswalpha(3C) を参照
iswgraph(3c)	iswalpha(3C) を参照
iswlower(3c)	iswalpha(3C) を参照
iswprint(3c)	iswalpha(3C) を参照
iswpunct(3c)	iswalpha(3C) を参照
iswspace(3c)	iswalpha(3C) を参照
iswupper(3c)	iswalpha(3C) を参照
iswxdigit(3c)	iswalpha(3C) を参照
isxdigit(3c)	ctype(3C) を参照
libcfgadm(3LIB)	構成管理のライブラリ
mblen(3C)	文字のバイト数を取得する
mbstowcs(3C)	文字列をワイド文字列に変換
mbtowc(3C)	複数バイト文字をワイド文字コードに変換
movenextch(3curses)	curs_alecompat(3CURSES) を参照
moveprevch(3curses)	curs_alecompat(3CURSES) を参照

<code>mvaddnwstr(3curses)</code>	<code> curs_addwstr(3CURSES)</code> を参照
<code>mvaddwch(3curses)</code>	<code> curs_addwch(3CURSES)</code> を参照
<code>mvaddwchnstr(3curses)</code>	<code> curs_addwchstr(3CURSES)</code> を参照
<code>mvaddwchstr(3curses)</code>	<code> curs_addwchstr(3CURSES)</code> を参照
<code>mvaddwstr(3curses)</code>	<code> curs_addwstr(3CURSES)</code> を参照
<code>mvgetnwstr(3curses)</code>	<code> curs_getwstr(3CURSES)</code> を参照
<code>mvgetwch(3curses)</code>	<code> curs_getwch(3CURSES)</code> を参照
<code>mvgetwstr(3curses)</code>	<code> curs_getwstr(3CURSES)</code> を参照
<code>mvinnwstr(3curses)</code>	<code> curs_inwstr(3CURSES)</code> を参照
<code>mvinsnwstr(3curses)</code>	<code> curs_inswstr(3CURSES)</code> を参照
<code>mvinswch(3curses)</code>	<code> curs_inswch(3CURSES)</code> を参照
<code>mvinswstr(3curses)</code>	<code> curs_inswstr(3CURSES)</code> を参照
<code>mvinwch(3curses)</code>	<code> curs_inwch(3CURSES)</code> を参照
<code>mvinwchnstr(3curses)</code>	<code> curs_inwchstr(3CURSES)</code> を参照
<code>mvinwchstr(3curses)</code>	<code> curs_inwchstr(3CURSES)</code> を参照
<code>mvinwstr(3curses)</code>	<code> curs_inwstr(3CURSES)</code> を参照
<code>mvwaddnwstr(3curses)</code>	<code> curs_addwstr(3CURSES)</code> を参照
<code>mvwaddwch(3curses)</code>	<code> curs_addwch(3CURSES)</code> を参照
<code>mvwaddwchnstr(3curses)</code>	<code> curs_addwchstr(3CURSES)</code> を参照
<code>mvwaddwchstr(3curses)</code>	<code> curs_addwchstr(3CURSES)</code> を参照
<code>mvwaddwstr(3curses)</code>	<code> curs_addwstr(3CURSES)</code> を参照
<code>mvwgetnwstr(3curses)</code>	<code> curs_getwstr(3CURSES)</code> を参照
<code>mvwgetwch(3curses)</code>	<code> curs_getwch(3CURSES)</code> を参照
<code>mvwgetwstr(3curses)</code>	<code> curs_getwstr(3CURSES)</code> を参照
<code>mvwinnwstr(3curses)</code>	<code> curs_inwstr(3CURSES)</code> を参照
<code>mvwinsnwstr(3curses)</code>	<code> curs_inswstr(3CURSES)</code> を参照
<code>mvwinswch(3curses)</code>	<code> curs_inswch(3CURSES)</code> を参照
<code>mvwinswstr(3curses)</code>	<code> curs_inswstr(3CURSES)</code> を参照
<code>mvwinwch(3curses)</code>	<code> curs_inwch(3CURSES)</code> を参照
<code>mvwinwchnstr(3curses)</code>	<code> curs_inwchstr(3CURSES)</code> を参照
<code>mvwinwchstr(3curses)</code>	<code> curs_inwchstr(3CURSES)</code> を参照

## Intro(3)

<code>mvwinwstr(3curses)</code>	<code> curs_inwstr(3CURSES)</code> を参照
<code>newpad(3curses)</code>	<code> curs_pad(3CURSES)</code> を参照
<code>ngettext(3c)</code>	<code> gettext(3C)</code> を参照
<code>pechochar(3curses)</code>	<code> curs_pad(3CURSES)</code> を参照
<code>pechowchar(3curses)</code>	<code> curs_pad(3CURSES)</code> を参照
<code>pnoutrefresh(3curses)</code>	<code> curs_pad(3CURSES)</code> を参照
<code>prefresh(3curses)</code>	<code> curs_pad(3CURSES)</code> を参照
<code>printf(3C)</code>	書式付き出力
<code>putwc(3c)</code>	<code> fputwc(3C)</code> を参照
<code>putwchar(3c)</code>	<code> fputwc(3C)</code> を参照
<code>putws(3C)</code>	ワイド文字列の EUC 文字への変換
<code>regexpr(3GEN)</code>	正規表現のコンパイルおよび一致ルーチン
<code>scanf(3C)</code>	書式付き入力の変換
<code>setlocale(3C)</code>	プログラムのロケールの変更および照会
<code>snprintf(3c)</code>	<code> printf(3C)</code> を参照
<code>sprintf(3c)</code>	<code> printf(3C)</code> を参照
<code>sscanf(3c)</code>	<code> scanf(3C)</code> を参照
<code>step(3gen)</code>	<code> regexpr(3GEN)</code> を参照
<code>strftime(3C)</code>	日付と時刻を文字列に変換
<code>subpad(3curses)</code>	<code> curs_pad(3CURSES)</code> を参照
<code>textdomain(3c)</code>	<code> gettext(3C)</code> を参照
<code>toascii(3C)</code>	整数を 7 ビット ASCII 文字へ変換
<code>_tolower(3C)</code>	大文字を小文字に変換
<code>tolower(3C)</code>	大文字を小文字に変換
<code>_toupper(3C)</code>	小文字を大文字に変換
<code>toupper(3C)</code>	小文字を大文字に変換
<code>towlower(3C)</code>	ワイド文字表現の大文字を小文字に変換
<code>towupper(3C)</code>	ワイド文字表現の小文字を大文字に変換
<code>ungetwc(3C)</code>	ワイド文字コードを入力ストリームに プッシュバック
<code>ungetwch(3curses)</code>	<code> curs_getwch(3CURSES)</code> を参照



<code>vfscanf(3c)</code>	<code>scanf(3C)</code> を参照
<code>vscanf(3c)</code>	<code>scanf(3C)</code> を参照
<code>vsscanf(3c)</code>	<code>scanf(3C)</code> を参照
<code>waddnwstr(3curses)</code>	<code>curl_addwstr(3CURSES)</code> を参照
<code>waddwch(3curses)</code>	<code>curl_addwch(3CURSES)</code> を参照
<code>waddwchnstr(3curses)</code>	<code>curl_addwchstr(3CURSES)</code> を参照
<code>waddwchstr(3curses)</code>	<code>curl_addwchstr(3CURSES)</code> を参照
<code>waddwstr(3curses)</code>	<code>curl_addwstr(3CURSES)</code> を参照
<code>wadjcurpos(3curses)</code>	<code>curl_alecompat(3CURSES)</code> を参照
<code>watof(3c)</code>	<code>wcstod(3C)</code> を参照
<code>watoi(3c)</code>	<code>wcstol(3C)</code> を参照
<code>watol(3c)</code>	<code>wcstol(3C)</code> を参照
<code>watoll(3c)</code>	<code>wcstol(3C)</code> を参照
<code>wscat(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wchr(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wscmp(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wscoll(3C)</code>	照合情報を使ってワイド文字列を比較
<code>wscopy(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wscspn(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wcsetno(3c)</code>	<code>cset(3C)</code> を参照
<code>wcsftime(3C)</code>	日付および時刻をワイド文字列に変換
<code>wcslen(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wcsncat(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wcsncmp(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wcsncpy(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wcspbrk(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wcsrchr(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wcsspn(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wcstod(3C)</code>	ワイド文字列を倍精度浮動小数点数に変換
<code>wcstok(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wcstol(3C)</code>	ワイド文字列を整数に変換

## Intro(3)

<code>wcstombs(3C)</code>	ワイド文字列を文字列に変換
<code>wcstoul(3C)</code>	ワイド文字列を符号なしロング整数に変換
<code>wcstring(3C)</code>	ワイド文字列の操作
<code>wcswcs(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wcswidth(3C)</code>	ワイド文字列のカラム数
<code>wcsxfrm(3C)</code>	ワイド文字列の変換
<code>wctomb(3C)</code>	ワイド文字コードを複数バイト文字に変換
<code>wctype(3C)</code>	文字クラスの定義
<code>wcwidth(3C)</code>	ワイド文字コードのカラム数
<code>wechowchar(3curses)</code>	<code>curs_addwch(3CURSES)</code> を参照
<code>wgetnwstr(3curses)</code>	<code>curs_getwstr(3CURSES)</code> を参照
<code>wgetwch(3curses)</code>	<code>curs_getwch(3CURSES)</code> を参照
<code>wgetwstr(3curses)</code>	<code>curs_getwstr(3CURSES)</code> を参照
<code>windex(3c)</code>	<code>wcstring(3C)</code> を参照
<code>winwstr(3curses)</code>	<code>curs_inwstr(3CURSES)</code> を参照
<code>winswstr(3curses)</code>	<code>curs_inswstr(3CURSES)</code> を参照
<code>winswch(3curses)</code>	<code>curs_inswch(3CURSES)</code> を参照
<code>winswstr(3curses)</code>	<code>curs_inswstr(3CURSES)</code> を参照
<code>winwch(3curses)</code>	<code>curs_inwch(3CURSES)</code> を参照
<code>winwchnstr(3curses)</code>	<code>curs_inwchstr(3CURSES)</code> を参照
<code>winwchstr(3curses)</code>	<code>curs_inwchstr(3CURSES)</code> を参照
<code>winwstr(3curses)</code>	<code>curs_inwstr(3CURSES)</code> を参照
<code>wmovenextch(3curses)</code>	<code>curs_alecompat(3CURSES)</code> を参照
<code>wmoveprevch(3curses)</code>	<code>curs_alecompat(3CURSES)</code> を参照
<code>wrindex(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wscasecmp(3c)</code>	<code>wstring(3C)</code> を参照
<code>wscat(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wschr(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wscmp(3c)</code>	<code>wcstring(3C)</code> を参照
<code>wscol(3c)</code>	<code>wstring(3C)</code> を参照
<code>wscoll(3c)</code>	<code>wscoll(3C)</code> を参照

wscopy(3c)	wcstring(3C) を参照
wscspn(3c)	wcstring(3C) を参照
wsdup(3c)	wstring(3C) を参照
wslen(3c)	wcstring(3C) を参照
wsncasecmp(3c)	wstring(3C) を参照
wsncat(3c)	wcstring(3C) を参照
wsncmp(3c)	wcstring(3C) を参照
wsncpy(3c)	wcstring(3C) を参照
wspbrk(3c)	wcstring(3C) を参照
wsprintf(3C)	書式付き出力変換
wsrchr(3c)	wcstring(3C) を参照
wsscanf(3C)	書式付き入力変換
wsspncpy(3c)	wcstring(3C) を参照
wstod(3c)	wcstod(3C) を参照
wstok(3c)	wcstring(3C) を参照
wstol(3c)	wcstol(3C) を参照
wstring(3C)	ワイド文字の文字列操作
wsxfrm(3c)	wcsxfrm(3C) を参照

## asctime(3C)

名前	strftime, cftime, asctime – 日付と時刻を文字列に変換												
形式	<pre>#include &lt;time.h&gt;  size_t <b>strftime</b>(const char *s, size_t <i>maxsize</i>, const char *<i>format</i>,                  const struct tm *<i>timeptr</i>);  int <b>cftime</b>(char *s, char *<i>format</i>, const time_t *<i>clock</i>);  int <b>asctime</b>(char *s, const char *<i>format</i>, const struct tm *<i>timeptr</i>);</pre>												
機能説明	<p>strftime()、asctime()、cftime() 関数は、<i>format</i> がポイントする文字列の制御に従って、バイトを <i>s</i> がポイントする配列に格納します。<i>format</i> 文字列は、0 個以上の変換仕様と通常文字からなります。変換仕様は、パーセント記号 (%) の後に、変換の内容を表す 1 つ以上の文字を伴います。終了 NULL バイトを含む通常文字すべては、そのまま <i>s</i> が示す配列に複写されます。オーバーラップしている複数のオブジェクト間で複写が行われると、変換の結果は予測できません。strftime() の場合は、<i>maxsize</i> を超えないバイト数の文字だけが配列に格納されます。</p> <p><i>format</i> が (char *)0 であれば、そのロケールのデフォルト形式が使用されます。strftime() の場合は、デフォルト形式は %c と同じになります。また、cftime() や asctime() の場合は、デフォルト形式は %c と同じになります。cftime() も asctime() も、まず環境変数の値 CFTIME を利用しようとしませんが、これが未定義または空であれば、デフォルト形式を使用します。</p> <p>各変換仕様は、以下の一覧に示したように、適切な文字に置き換えられます。適切な文字は、プログラムのロケールの LC_TIME カテゴリと、strftime() と asctime() の <i>timeptr</i> がポイントする構造体に入っている値、cftime() の <i>clock</i> で表されている時間で決まります。</p> <table border="0"> <tr><td>%%</td><td>% と同じ</td></tr> <tr><td>%a</td><td>省略形式の曜日名 (ロケール固有)</td></tr> <tr><td>%A</td><td>完全形式の曜日名 (ロケール固有)</td></tr> <tr><td>%b</td><td>省略形式の月名 (ロケール固有)</td></tr> <tr><td>%B</td><td>完全形式の月名 (ロケール固有)</td></tr> <tr><td>%c</td><td>日時の表現 (ロケール固有)</td></tr> </table>	%%	% と同じ	%a	省略形式の曜日名 (ロケール固有)	%A	完全形式の曜日名 (ロケール固有)	%b	省略形式の月名 (ロケール固有)	%B	完全形式の月名 (ロケール固有)	%c	日時の表現 (ロケール固有)
%%	% と同じ												
%a	省略形式の曜日名 (ロケール固有)												
%A	完全形式の曜日名 (ロケール固有)												
%b	省略形式の月名 (ロケール固有)												
%B	完全形式の月名 (ロケール固有)												
%c	日時の表現 (ロケール固有)												
デフォルト	%C date(1) が出力する形式の日時の表現												
規格準拠	<table border="0"> <tr><td>%C</td><td>世紀の番号。年を 100 で除算して余りを切り捨てて得られる 1 から 99 までの整数。1 桁の場合は前に 0 が付加される (standards(5) を参照)</td></tr> <tr><td>%d</td><td>日 (1 から 31)。1 桁の場合は前に 0 が付加される</td></tr> <tr><td>%D</td><td>%m/%d/%y 形式で表した日付</td></tr> <tr><td>%e</td><td>日 (1 から 31)。1 桁の場合は前に空白文字が付加される</td></tr> <tr><td>%g</td><td>週をベースにした西暦年の下 2 桁 (00 から 99)</td></tr> <tr><td>%G</td><td>週をベースにした西暦年の完全表示 (0000 から 9999)</td></tr> </table>	%C	世紀の番号。年を 100 で除算して余りを切り捨てて得られる 1 から 99 までの整数。1 桁の場合は前に 0 が付加される (standards(5) を参照)	%d	日 (1 から 31)。1 桁の場合は前に 0 が付加される	%D	%m/%d/%y 形式で表した日付	%e	日 (1 から 31)。1 桁の場合は前に空白文字が付加される	%g	週をベースにした西暦年の下 2 桁 (00 から 99)	%G	週をベースにした西暦年の完全表示 (0000 から 9999)
%C	世紀の番号。年を 100 で除算して余りを切り捨てて得られる 1 から 99 までの整数。1 桁の場合は前に 0 が付加される (standards(5) を参照)												
%d	日 (1 から 31)。1 桁の場合は前に 0 が付加される												
%D	%m/%d/%y 形式で表した日付												
%e	日 (1 から 31)。1 桁の場合は前に空白文字が付加される												
%g	週をベースにした西暦年の下 2 桁 (00 から 99)												
%G	週をベースにした西暦年の完全表示 (0000 から 9999)												

%h	省略形式の月名 (ロケール固有)
%H	24 時間表示で表した時 (0 から 23)。1 桁の場合は前に 0 が付加される
%I	12 時間表示で表した時 (1 から 12)。1 桁の場合は前に 0 が付加される
%j	年の通算日 (1 から 366)。3 桁未満の場合は前に 0 が付加される
%k	24 時間表示で表した時 (0 から 23)。1 桁の場合は前に空白文字が付加される
%l	12 時間表示で表した時 (1 から 12)。1 桁の場合は前に空白文字が付加される
%m	月 (1 から 12)。1 桁の場合は前に 0 が付加される
%M	分 (00 から 59)。先行する 0 は任意
%n	復帰改行を挿入
%p	午前または午後を表す値 (ロケール固有)
%r	%p を伴った、12 時間表示形式で表した時刻
%R	%H:%M 形式で表した時刻
%S	秒 (00 から 61)。値の範囲が (00 から 59) ではなく (00 から 61) であるのは、うるう秒および二重うるう秒に対処するため
%t	タブを挿入
%T	%H:%M:%S 形式で表した時刻
%u	曜日を表す番号 (1 から 7)。日曜日が 1 (「注意事項」を参照)
%U	その年の何週目かを表す数値 (00 から 53)。第 1 週はその年の最初の日曜日 から始まるとする
%V	ISO 8601 で何週目かを表す数値 (01 から 53)。ISO 8601 の週をベースとするシステムでは、週は月曜日から始まり、1 月 4 日およびその年の最初の木曜日の両方を含む週が第 1 週となる。その年の最初の月曜日が 1 月 1 日、2 日、3 日、または 4 日の場合には、その週は前年の最終週の一部になる (「注意事項」を参照)
%w	曜日を表す番号 (0 から 6)。日曜日が 0
%W	その年の何週目かを表す数値 (00 から 53)。第 1 週はその年の最初の月曜日から始まるとする
%x	適切な日付の表現 (ロケール固有)
%X	適切な時間の表現 (ロケール固有)
%y	西暦年の下 2 桁 (00 から 99)
%Y	西暦年の完全表示 (たとえば 1993)
%Z	タイムゾーン名 (タイムゾーンがない場合には文字なし)

## asctime(3C)

上記の変換仕様または後述する変更指定のどれにも該当しない指定を記述すると、変換結果は未定義で 0 が返されます。

%U と %W (このほか、後述の %OU と %OW) とでは、何曜日を週の第 1 日目とするかが異なっています。たとえば、1 月の第 1 週目 (週番号 1) は、%U では日曜日から、%W では月曜日から始まります。また、週番号 0 には、%U では 1 月の第 1 日曜日より前の日が、%W では 1 月の第 1 日曜日より前の日が含まれています。

### 変換仕様の変更

上記の変換仕様に、変更を表す文字 E または O を付加することができます。このようにすると、代替形式または代替指定で値を得ることが可能です。希望した代替形式または代替指定が現ロケール中に存在していなければ、変更を示す文字を指定しなかった場合の形式で値が得られます。

%Ec	適切な日付および時刻の代替形式 (ロケール固有)
%EC	代替表示形式として指定されている年号 (ロケール固有)
%Eg	代替表示形式として指定されている年号 (%EC) に対応した週をベースとした年 (ロケール固有)
%EG	完全形式の週をベースとした代替年表示
%Ex	日付の代替表示形式 (ロケール固有)
%EX	時刻の代替表示形式 (ロケール固有)
%Ey	代替表示形式として指定されている年号 (%EC) に対応した年 (ロケール固有)
%EY	完全形式の代替年表示
%Od	日をロケール固有の代替数値記号で表す
%Oe	%Od と同じ
%Og	ロケール固有の週をベースとした代替表示の年 (%c に対応) の値を、ロケール固有の代替数値記号で表す
%OH	24 時間表示での時をロケール固有の代替数値記号で表す
%OI	12 時間表示での時をロケール固有の代替数値記号で表す
%Om	月をロケール固有の代替数値記号で表す
%OM	分をロケール固有の代替数値記号で表す
%OS	秒をロケール固有の代替数値記号で表す
%Ou	週日を示す値をロケール固有の代替数値記号にある数字で表す
%OU	その年の何週目かをロケール固有の代替数値記号で表す。週は日曜日から始まるとする
%Ow	曜日を示す値をロケール固有の代替数値記号で表す。日曜日を 0 とする
%OW	その年の何週目かをロケール固有の代替数値記号で表す。週は月曜日から始まるとする

出力言語の選択	<p>%Oy      ロケール固有の代替表示の年 (%c に対応) の値を、ロケール固有の代替数値記号で表す</p> <p>デフォルトでは、<code>strftime()</code>、<code>cftime()</code>、<code>ascftime()</code> は米国英語で表示されま す。<code>setlocale()</code> で <code>LC_TIME</code> カテゴリのロケールを設定すると、<code>strftime()</code>、 <code>cftime()</code>、<code>ascftime()</code> の実行結果を特定の言語で出力できるようになります。</p>						
タイムゾーン	<p><code>tzset(3C)</code> を呼び出した場合と同じタイムゾーン情報を使用します。</p>						
戻り値	<p><code>strftime()</code>、<code>cftime()</code>、<code>ascftime()</code> 関数は、<code>s</code> が示す配列に書き出した文字数 を返します。これには終了 <code>NULL</code> 文字は含まれません。生成された文字の合計数 (終 了 <code>NULL</code> 文字を含む) が <code>maxsize</code> を超えていると、<code>strftime()</code> は、0 を返し、配列 の内容は未確定になります。</p>						
使用例	<p>例 1 An example of the <code>strftime()</code> function.</p> <p>ここでは、POSIX ロケールの <code>strftime()</code> の使用例を示します。<code>tmpr</code> が指す構造体 に 1986 年 8 月 28 日木曜日 12 時 44 分 36 秒に対応する値が入っている場合、文字列 <code>str</code> の内容がどうなるかを示します。</p> <pre>strftime (str, strsize, "%A %b %d %j", tmpr)</pre> <p>これにより、<code>str</code> の内容は "Thursday Aug 28 240" になります。</p>						
属性	<p>次の属性については <code>attributes(5)</code> のマニュアルページを参照してください。</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">属性タイプ</th> <th style="text-align: center;">属性値</th> </tr> </thead> <tbody> <tr> <td>MT レベル</td> <td>MT-safe</td> </tr> <tr> <td>CSI</td> <td>対応済み</td> </tr> </tbody> </table>	属性タイプ	属性値	MT レベル	MT-safe	CSI	対応済み
属性タイプ	属性値						
MT レベル	MT-safe						
CSI	対応済み						
関連項目	<p><code>date(1)</code>, <code>ctime(3C)</code>, <code>mktime(3C)</code>, <code>setlocale(3C)</code>, <code>strptime(3C)</code>, <code>tzset(3C)</code>, <code>TIMEZONE(4)</code>, <code>zoneinfo(4)</code>, <code>attributes(5)</code>, <code>environ(5)</code>, <code>standards(5)</code></p>						
注意事項	<p>Solaris 7 では、%V の変換仕様が変わりました。この変更は、現時点で公表されている ISO C9x 標準の決定に基づいています。以前の仕様では、1 月 1 日を含む週に、4 日未 満の日数の新しい年が含まれている場合、その週は前年の第 53 週となりましたが、 ISO C9x 標準委員会によって、この仕様が正しくないと判断されました。</p> <p>%g、%G、%Eg、%EG、%Og の変換仕様が Solaris 7 に追加されました。この変更は、現 時点では公表されている ISO C9x 標準の決定に基づいています。これらの仕様は ISO C9x 標準の決定によって変更される可能性があり、変更前の仕様との互換性が保たれ ないこともあります。</p> <p>Solaris 8 リリースでは、%u の変換仕様に変更されました。この変更は、XPG4 仕様に 基づいたものです。</p>						

## ascftime(3C)

%Z 指示子と zoneinfo タイムゾーンを使用している場合に、入力日時が 20:45:52 UTC, December 13, 1901 から 03:14:07 UTC, January 19, 2038 までの範囲外にある場合、タイムゾーン名が正しくないことがあります。



名前	gettext, dgettext, dcgettext, ngettext, dngettext, dcngettext, textdomain, bindtextdomain, bind_textdomain_codeset – メッセージ処理関数
形式	
Solaris および GNU 互換	<pre>#include &lt;libintl.h&gt; char *gettext(const char *msgid); char *dgettext(const char *domainname, const char *msgid); char *textdomain(const char *domainname); char *bindtextdomain(const char *domainname, const char *dirname); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcgettext(const char *domainname, const char *msgid, int category);</pre>
GNU 互換	<pre>#include &lt;libintl.h&gt; char *ngettext(const char *msgid1, const char *msgid2, unsigned long int n); char *dngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n); char *bind_textdomain_codeset(const char *domainname, const char *codeset); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n, int category);</pre>
機能説明	<p>gettext(), dgettext(), dcgettext() 関数は、特定のドメイン内において、また、現在のロケールにおいて、指定の <i>msgid</i> 引数に基づいたターゲット文字列を取得します。gettext(), dgettext(), dcgettext() から返される文字列の長さは、これらの関数が呼び出されるまでは未定です。<i>msgid</i> 引数は、NULL で終わる文字列です。</p> <p>ngettext(), dngettext(), および dcngettext() 関数はそれぞれ、gettext(), dgettext(), および dcgettext() と同等ですが、GNU 互換メッセージカタログだけを処理することができ、複数形の処理方法が異なります。メッセージ文字列を検索するとき、ngettext(), dngettext(), および dcngettext() 関数は <i>msgid1</i> 引数を検索鍵として使用し、<i>n</i> 引数で複数形を決定します。メッセージカタログが見つからない場合、<i>n</i> == 1 であれば <i>msgid1</i> が返され、そうでなければ <i>msgid2</i> が返されます。</p> <p>環境変数 NLSPATH (environ(5) を参照) が、最初に LC_MESSAGES カタログの位置として検索されます。現在のロケールの LC_MESSAGES カテゴリの設定によって、gettext() および dgettext() が文字列取得に使用するロケールが決まります。</p>

## bindtextdomain(3C)

*category* 引数は `dcgettext()` の使用するロケールを決めます。NLSPATH が定義されておらず、現在のロケールが C ロケールの場合、`gettext()`、`gettext()`、`dcgettext()` は、渡されたメッセージ文字列をそのまま返します。C 以外のロケールでは、NLSPATH が定義されていない場合、あるいは、NLSPATH で指定したコンポーネントでメッセージカタログが見つからない場合、メッセージカタログの検索には、次の段落で説明するスキームが使用されます。

LANGUAGE 環境変数は、使用する GNU 互換メッセージカタログを決定するために調べられます。LANGUAGE の値はコロン (:) で区切られたロケール名のリストです。LANGUAGE が定義されている場合、リストに指定されている順番で各ロケールが検索されます。そして、GNU 互換メッセージカタログが見つかった場合、そのメッセージが返されます。GNU 互換メッセージが見つかったが、対応する *msgid* が見つからない場合、*msgid* 文字列が返されます。LANGUAGE が定義されていない場合、あるいは、LANGUAGE の処理中に Solaris メッセージカタログが見つかったが GNU 互換メッセージカタログが見つからない場合、*dirname/locale/category/domainname.mo* というパス名でメッセージカタログが検索されます。ここで、*dirname* は `bindtextdomain()` で指定されたディレクトリ、*locale* はロケール名、*category* は LC\_MESSAGES または LC\_XXX のどちらかです。LC\_MESSAGES は `gettext()`、`dgettext()`、`ngettext()`、または `dngettext()` を呼び出した場合に使用されます。LC\_XXX は `dcgettext()` または `dcngettext()` を呼び出した場合に使用され、XXX 部分は *category* 引数で指定したロケールカテゴリ名と同じになります。

`gettext()` と `ngettext()` の場合、使用されるドメインは、最後に行なった `textdomain()` の有効な呼び出しによって設定されます。`textdomain()` の有効な呼び出しが行われていない場合は、デフォルト時のドメイン (`messages`) が使用されます。

`dgettext()`、`dngettext()`、`dcngettext()`、および `dcgettext()` の場合、使用されるドメインは *domainname* 引数で指定されます。この *domainname* 引数は、構文と意味において、`textdomain()` の *domainname* 引数と同等ですが、このドメインの選択が `dgettext()`、`dcgettext()`、`dngettext()` および `dcngettext()` 関数の呼び出し中のみ有効であるという点で異なります。

`textdomain()` 関数は、アクティブな LC\_MESSAGES ロケールカテゴリの現在のドメイン名の設定または照会を行います。*domainname* 引数は、正当なファイル名に使用できる文字のみからなる、NULL で終わる文字列です。

*domainname* 引数は、システム内の、一意のドメイン名です。1つのシステム内に、同一ドメインのバージョンが複数存在する場合は、`bindtextdomain()` を使用して名前空間の衝突を回避することができます。`textdomain()` を呼び出さない場合には、デフォルト時のドメインが選択されます。最後に行なった `textdomain()` の有効な呼び出しによるドメインの設定は、このあとに行う `setlocale(3C)` および `gettext()` の呼び出しにおいても有効です。

*domainname* 引数は、現在アクティブな、LC\_MESSAGES ロケールに適用されます。

*domainname* に NULL ポインタを設定して `textdomain()` を呼び出すことによって、ドメインの現在の状態に影響することなく、ドメインの現在の設定を照会することができます。*domainname* 引数に NULL 文字列を設定して `textdomain()` を呼び出すと、ドメインの設定は、デフォルト時のドメイン (`messages`) になります。

`bindtextdomain()` 関数は、メッセージドメインのパス部分 *domainname* と、*dirname* に入っている値を結合します。*domainname* が空文字列ではなく、まだ結合されていない場合には、`bindtextdomain()` は、*domainname* と *dirname* を結合します。

*domainname* が空文字列ではなく、すでに結合されている場合には、`bindtextdomain()` は、事前に行われていた結合部分を *dirname* に置き換えます。*dirname* 引数に使用できるのは、`gettext()`、`dgettext()`、または `dcgettext()` が呼び出されたときに解決される、絶対パス名または相対パス名です。*domainname* が NULL ポインタまたは空文字列の場合は、`bindtextdomain()` は NULL を返します。ユーザー定義のドメイン名は、`SYS_` で始めることはできません。この文字列で始まるドメイン名は、システムで使用するために予約されています。

`bind_textdomain_codeset()` 関数を使用すると、ドメイン *domainname* のメッセージカタログ用の出力コードセットを指定できます。*codeset* 引数は、`iconv_open(3C)` 関数に対して使用可能な有効なコードセット名であるか、空ポインタである必要があります。*codeset* 引数が空ポインタの場合、`bind_textdomain_codeset()` はドメイン *domainname* 用に現在選択されているコードセットを返します。コードセットが選択されていない場合、`bind_textdomain_codeset()` は空ポインタを返します。`bind_textdomain_codeset()` 関数は複数回使用できます。同じ *domainname* 引数を指定して `bind_textdomain_codeset()` 関数を複数回使用した場合、先行する呼び出しで行われた設定は後続の呼び出しによって無効にされます。`bind_textdomain_codeset()` 関数は、選択されたコードセット名を含む文字列へのポインタを返します。この文字列は `bind_textdomain_codeset()` 関数に内部的に割り当てられているため、ユーザーが変更してはなりません。

戻り値

`gettext()`、`dgettext()`、および `dcgettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*msgid* 文字列を返します。

`ngettext()`、`dngettext()`、および `dcngettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*n* == 1 であれば、*msgid1* を返します。そうでなければ、*msgid2* を返します。

`gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` が返す文字列の個々のバイトには、空文字以外のあらゆる値が入ります。*msgid* が NULL ポインタである場合の戻り値は未定義です。返される文字列は、プログラムで修正してはならず、この後で `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` または `setlocale(3C)` を呼び出すことにより、無効になることがあります。`dgettext()`、`dcgettext()`、`dngettext()`、または `dcngettext()` に対する *domainname* 引数が NULL ポインタである場合、現在 `textdomain()` によってバインドされているドメインが使用されます。

## bindtextdomain(3C)

`textdomain()` の正常な戻り値は、ドメインの現在の設定が入っている文字列へのポインタです。`domainname` が NULL ポインタの場合は、`textdomain()` は現在のドメインが入っている文字列へのポインタを返します。`textdomain()` を先に呼び出していない場合に `domainname` を空文字列にすると、デフォルト時のドメイン名が返されます。デフォルト時のドメイン名は、`messages` です。`textdomain()` が失敗した場合、NULL ポインタが返されます。

`bindtextdomain()` からの戻り値は、`dirname` が入った NULL で終わる文字列です。`dirname` が NULL の場合は、`domainname` と結び付けられたディレクトリバインディングです。何も結合されていない場合には、デフォルトの戻り値は `/usr/lib/locale` になります。`domainname` が NULL ポインタまたは空文字列の場合、`bindtextdomain()` は何も行わず、NULL ポインタを返します。返された文字列を呼び出し側で変更してはなりません。`bindtextdomain()` が失敗した場合、NULL ポインタが返されます。

使用法 これらのルーチンでは、メッセージの長さに制限はありません。ただし、テキスト `domainname` は、TEXTDOMAINMAX (256) バイト以内に制限されています。

ロケール変更のために `setlocale(3C)` が呼び出されていない限り、マルチスレッドアプリケーションにおいて `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()`、`textdomain()`、および `bindtextdomain()` 関数を安全に使うことができます。

`gettext()`、`dgettext()`、`dcgettext()`、`textdomain()`、および `bindtextdomain()` 関数は Solaris メッセージカタログと GNU 互換メッセージカタログの両方を処理できます。`ngettext()`、`dngettext()`、`dcngettext()`、および `bind_textdomain_codeset()` 関数は GNU 互換メッセージカタログだけを処理できます。Solaris メッセージカタログと GNU 互換メッセージカタログの詳細については、`msgfmt(1)` のマニュアルページを参照してください。

## ファイル

`/usr/lib/locale`  
メッセージ・ドメイン・ファイルのデフォルトパス部分

`/usr/lib/locale/locale/LC_MESSAGES/domainname.mo`  
言語 `locale` および `domainname` 用メッセージの入っているファイルのシステムデフォルト位置

`/usr/lib/locale/locale/LC_XXX/domainname.mo`  
`LC_XXX` が `LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` である場合に、`dcgettext()` 呼び出し用の言語 `locale` および `domainname` メッセージの入っているファイルのシステムデフォルト位置

`dirname/locale/LC_MESSAGES/domainname.mo`  
`bindtextdomain()` の正常な呼び出し後の、ドメイン `domainname` およびパス部分 `dirname` 用メッセージが入っているファイルの位置

`dirname/locale/LC_XXX/domainname.mo`  
`bindtextdomain()` の正常な呼び出し後の `LC_XXX` が、`LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` のいずれかである場合 `dcgettext()` 呼び出し用のドメイン `domainname`、言語

## bindtextdomain(3C)

*locale*、およびパス部分 *dirname* メッセージが入っているファイルの位置

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 `msgfmt(1)`, `xgettext(1)`, `iconv_open(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

## bind\_textdomain\_codeset(3C)

名前	gettext, dgettext, dcgettext, ngettext, dngettext, dcngettext, textdomain, bindtextdomain, bind_textdomain_codeset – メッセージ処理関数
形式	
Solaris および GNU 互換	<pre>#include &lt;libintl.h&gt; char *gettext(const char *msgid); char *dgettext(const char *domainname, const char *msgid); char *textdomain(const char *domainname); char *bindtextdomain(const char *domainname, const char *dirname); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcgettext(const char *domainname, const char *msgid, int category);</pre>
GNU 互換	<pre>#include &lt;libintl.h&gt; char *ngettext(const char *msgid1, const char *msgid2, unsigned long int n); char *dngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n); char *bind_textdomain_codeset(const char *domainname, const char *codeset); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n, int category);</pre>
機能説明	<p>gettext(), dgettext(), dcgettext() 関数は、特定のドメイン内において、また、現在のロケールにおいて、指定の <i>msgid</i> 引数に基づいたターゲット文字列を取得します。gettext(), dgettext(), dcgettext() から返される文字列の長さは、これらの関数が呼び出されるまでは未定です。<i>msgid</i> 引数は、NULL で終わる文字列です。</p> <p>ngettext(), dngettext(), および dcngettext() 関数はそれぞれ、gettext(), dgettext(), および dcgettext() と同等ですが、GNU 互換メッセージカタログだけを処理することができ、複数形の処理方法が異なります。メッセージ文字列を検索するとき、ngettext(), dngettext(), および dcngettext() 関数は <i>msgid1</i> 引数を検索鍵として使用し、<i>n</i> 引数で複数形を決定します。メッセージカタログが見つからない場合、<i>n</i> == 1 であれば <i>msgid1</i> が返され、そうでなければ <i>msgid2</i> が返されます。</p> <p>環境変数 NLSPATH (environ(5) を参照) が、最初に LC_MESSAGES カタログの位置として検索されます。現在のロケールの LC_MESSAGES カテゴリの設定によって、gettext() および dgettext() が文字列取得に使用するロケールが決まります。</p>

*category* 引数は `dcgettext()` の使用するロケールを決めます。NLSPATH が定義されておらず、現在のロケールが C ロケールの場合、`gettext()`、`gettext()`、`dcgettext()` は、渡されたメッセージ文字列をそのまま返します。C 以外のロケールでは、NLSPATH が定義されていない場合、あるいは、NLSPATH で指定したコンポーネントでメッセージカタログが見つからない場合、メッセージカタログの検索には、次の段落で説明するスキームが使用されます。

LANGUAGE 環境変数は、使用する GNU 互換メッセージカタログを決定するために調べられます。LANGUAGE の値はコロン (:) で区切られたロケール名のリストです。LANGUAGE が定義されている場合、リストに指定されている順番で各ロケールが検索されます。そして、GNU 互換メッセージカタログが見つかった場合、そのメッセージが返されます。GNU 互換メッセージが見つかったが、対応する *msgid* が見つからない場合、*msgid* 文字列が返されます。LANGUAGE が定義されていない場合、あるいは、LANGUAGE の処理中に Solaris メッセージカタログが見つかったが GNU 互換メッセージカタログが見つからない場合、*dirname/locale/category/domainname.mo* というパス名でメッセージカタログが検索されます。ここで、*dirname* は `bindtextdomain()` で指定されたディレクトリ、*locale* はロケール名、*category* は LC\_MESSAGES または LC\_XXX のどちらかです。LC\_MESSAGES は `gettext()`、`dgettext()`、`ngettext()`、または `dngettext()` を呼び出した場合に使用されます。LC\_XXX は `dcgettext()` または `dcngettext()` を呼び出した場合に使用され、XXX 部分は *category* 引数で指定したロケールカテゴリ名と同じになります。

`gettext()` と `ngettext()` の場合、使用されるドメインは、最後に行なった `textdomain()` の有効な呼び出しによって設定されます。`textdomain()` の有効な呼び出しが行われていない場合は、デフォルト時のドメイン (`messages`) が使用されます。

`dgettext()`、`dngettext()`、`dcngettext()`、および `dcgettext()` の場合、使用されるドメインは *domainname* 引数で指定されます。この *domainname* 引数は、構文と意味において、`textdomain()` の *domainname* 引数と同等ですが、このドメインの選択が `dgettext()`、`dcgettext()`、`dngettext()` および `dcngettext()` 関数の呼び出し中にのみ有効であるという点で異なります。

`textdomain()` 関数は、アクティブな LC\_MESSAGES ロケールカテゴリの現在のドメイン名の設定 または照会を行います。*domainname* 引数は、正当なファイル名に使用できる文字のみからなる、NULL で終わる文字列です。

*domainname* 引数は、システム内の、一意のドメイン名です。1つのシステム内に、同一ドメインのバージョンが複数存在する場合は、`bindtextdomain()` を使用して名前空間の衝突を回避することができます。`textdomain()` を呼び出さない場合には、デフォルト時のドメインが選択されます。最後に行なった `textdomain()` の有効な呼び出しによるドメインの設定は、このあとに行う `setlocale(3C)` および `gettext()` の呼び出しにおいても有効です。

*domainname* 引数は、現在アクティブな、LC\_MESSAGES ロケールに適用されます。

## bind\_textdomain\_codeset(3C)

*domainname* に NULL ポインタを設定して `textdomain()` を呼び出すことによって、ドメインの現在の状態に影響することなく、ドメインの現在の設定を照会することができます。*domainname* 引数に NULL 文字列を設定して `textdomain()` を呼び出すと、ドメインの設定は、デフォルト時のドメイン (`messages`) になります。

`bindtextdomain()` 関数は、メッセージドメインのパス部分 *domainname* と、*dirname* に入っている値を結合します。*domainname* が空文字列ではなく、まだ結合されていない場合には、`bindtextdomain()` は、*domainname* と *dirname* を結合します。

*domainname* が空文字列ではなく、すでに結合されている場合には、`bindtextdomain()` は、事前に行われていた結合部分を *dirname* に置き換えます。*dirname* 引数に使用できるのは、`gettext()`、`dgettext()`、または `dcgettext()` が呼び出されたときに解決される、絶対パス名または相対パス名です。*domainname* が NULL ポインタまたは空文字列の場合は、`bindtextdomain()` は NULL を返します。ユーザー定義のドメイン名は、`sys_` で始めることはできません。この文字列で始まるドメイン名は、システムで使用するために予約されています。

`bind_textdomain_codeset()` 関数を使用すると、ドメイン *domainname* のメッセージカタログ用の出力コードセットを指定できます。*codeset* 引数は、`iconv_open(3C)` 関数に対して使用可能な有効なコードセット名であるか、空ポインタである必要があります。*codeset* 引数が空ポインタの場合、`bind_textdomain_codeset()` はドメイン *domainname* 用に現在選択されているコードセットを返します。コードセットが選択されていない場合、`bind_textdomain_codeset()` は空ポインタを返します。`bind_textdomain_codeset()` 関数は複数回使用できます。同じ *domainname* 引数を指定して `bind_textdomain_codeset()` 関数を複数回使用した場合、先行する呼び出しで行われた設定は後続の呼び出しによって無効にされます。`bind_textdomain_codeset()` 関数は、選択されたコードセット名を含む文字列へのポインタを返します。この文字列は `bind_textdomain_codeset()` 関数に内部的に割り当てられているため、ユーザーが変更してはなりません。

戻り値 `gettext()`、`dgettext()`、および `dcgettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*msgid* 文字列を返します。

`ngettext()`、`dngettext()`、および `dcngettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*n* == 1 であれば、*msgid1* を返します。そうでなければ、*msgid2* を返します。

`gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` が返す文字列の個々のバイトには、空文字以外のあらゆる値が入ります。*msgid* が NULL ポインタである場合の戻り値は未定義です。返される文字列は、プログラムで修正してはならず、この後で `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` または `setlocale(3C)` を呼び出すことにより、無効になることがあります。`dgettext()`、`dcgettext()`、`dngettext()`、または `dcngettext()` に対する *domainname* 引数が NULL ポインタである場合、現在 `textdomain()` によってバインドされているドメインが使用されます。



## bind\_textdomain\_codeset(3C)

textdomain() の正常な戻り値は、ドメインの現在の設定が入っている文字列へのポインタです。domainname が NULL ポインタの場合は、textdomain() は現在のドメインが入っている文字列へのポインタを返します。textdomain() を先に呼び出していない場合に domainname を空文字列にすると、デフォルト時のドメイン名が返されます。デフォルト時のドメイン名は、messages です。textdomain() が失敗した場合、NULL ポインタが返されます。

bindtextdomain() からの戻り値は、dirname が入った NULL で終わる文字列です。dirname が NULL の場合は、domainname と結び付けられたディレクトリバインディングです。何も結合されていない場合には、デフォルトの戻り値は /usr/lib/locale になります。domainname が NULL ポインタまたは空文字列の場合、bindtextdomain() は何も行わず、NULL ポインタを返します。返された文字列を呼び出し側で変更してはなりません。bindtextdomain() が失敗した場合、NULL ポインタが返されます。

使用法 これらのルーチンでは、メッセージの長さに制限はありません。ただし、テキスト domainname は、TEXTDOMAINMAX (256) バイト以内に制限されています。

ロケール変更のために setlocale(3C) が呼び出されていない限り、マルチスレッドアプリケーションにおいて gettext(), dgettext(), dcgettext(), ngettext(), dngettext(), dcngettext(), textdomain(), および bindtextdomain() 関数を安全に使うことができます。

gettext(), dgettext(), dcgettext(), textdomain(), および bindtextdomain() 関数は Solaris メッセージカタログと GNU 互換メッセージカタログの両方を処理できます。ngettext(), dngettext(), dcngettext(), および bind\_textdomain\_codeset() 関数は GNU 互換メッセージカタログだけを処理できます。Solaris メッセージカタログと GNU 互換メッセージカタログの詳細については、msgfmt(1) のマニュアルページを参照してください。

ファイル

/usr/lib/locale

メッセージ・ドメイン・ファイルのデフォルトパス部分

/usr/lib/locale/locale/LC\_MESSAGES/domainname.mo

言語 locale および domainname 用メッセージの入っているファイルのシステムデフォルト位置

/usr/lib/locale/locale/LC\_XXX/domainname.mo

LC\_XXX が LC\_CTYPE、LC\_NUMERIC、LC\_TIME、LC\_COLLATE、LC\_MONETARY、または LC\_MESSAGES である場合に、dcgettext() 呼び出し用の言語 locale および domainname メッセージの入っているファイルのシステムデフォルト位置

dirname/locale/LC\_MESSAGES/domainname.mo

bindtextdomain() の正常な呼び出し後の、ドメイン domainname およびパス部分 dirname 用メッセージが入っているファイルの位置

dirname/locale/LC\_XXX/domainname.mo

bindtextdomain() の正常な呼び出し後の LC\_XXX が、LC\_CTYPE、LC\_NUMERIC、LC\_TIME、LC\_COLLATE、LC\_MONETARY、または LC\_MESSAGES のいずれかである場合 dcgettext() 呼び出し用のドメイン domainname、言語

## bind\_textdomain\_codeset(3C)

属性 *locale*、およびパス部分 *dirname* メッセージが入っているファイルの位置  
次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 `msgfmt(1)`, `xgettext(1)`, `iconv_open(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

名前	strftime, cftime, ascftime – 日付と時刻を文字列に変換
形式	<pre>#include &lt;time.h&gt;  size_t <b>strftime</b>(const char *s, size_t maxsize, const char *format,                  const struct tm *timeptr);  int <b>cftime</b>(char *s, char *format, const time_t *clock);  int <b>ascftime</b>(char *s, const char *format, const struct tm *timeptr);</pre>
機能説明	<p>strftime(), ascftime(), cftime() 関数は、<i>format</i> がポイントする文字列の制御に従って、バイトを <i>s</i> がポイントする配列に格納します。<i>format</i> 文字列は、0 個以上の変換仕様と通常文字からなります。変換仕様は、パーセント記号 (%) の後に、変換の内容を表す 1 つ以上の文字を伴います。終了 NULL バイトを含む通常文字すべては、そのまま <i>s</i> が示す配列に複写されます。オーバーラップしている複数のオブジェクト間で複写が行われると、変換の結果は予測できません。strftime() の場合は、<i>maxsize</i> を超えないバイト数の文字だけが配列に格納されます。</p> <p><i>format</i> が (char *)0 であれば、そのロケールのデフォルト形式が使用されます。strftime() の場合は、デフォルト形式は %c と同じになります。また、cftime() や ascftime() の場合は、デフォルト形式は %C と同じになります。cftime() も ascftime() も、まず環境変数の値 CFTIME を利用しようとしませんが、これが未定義または空であれば、デフォルト形式を使用します。</p> <p>各変換仕様は、以下の一覧に示したように、適切な文字に置き換えられます。適切な文字は、プログラムのロケールの LC_TIME カテゴリと、strftime() と ascftime() の <i>timeptr</i> がポイントする構造体に入っている値、cftime() の <i>clock</i> で表されている時間で決まります。</p>
デフォルト規格準拠	<pre>%%      % と同じ %a       省略形式の曜日名 (ロケール固有) %A       完全形式の曜日名 (ロケール固有) %b       省略形式の月名 (ロケール固有) %B       完全形式の月名 (ロケール固有) %c       日時の表現 (ロケール固有) %C       date(1) が出力する形式の日時の表現 %C       世紀の番号。年を 100 で除算して余りを切り捨てて得られる 1 から 99 までの整数。1 桁の場合は前に 0 が付加される (standards(5) を参照) %d       日 (1 から 31)。1 桁の場合は前に 0 が付加される %D       %m/%d/%y 形式で表した日付 %e       日 (1 から 31)。1 桁の場合は前に空白文字が付加される %g       週をベースにした西暦年の下 2 桁 (00 から 99) %G       週をベースにした西暦年の完全表示 (0000 から 9999)</pre>

## cftime(3C)

%h	省略形式の月名 (ロケール固有)
%H	24 時間表示で表した時 (0 から 23)。1 桁の場合は前に 0 が付加される
%I	12 時間表示で表した時 (1 から 12)。1 桁の場合は前に 0 が付加される
%j	年の通算日 (1 から 366)。3 桁未満の場合は前に 0 が付加される
%k	24 時間表示で表した時 (0 から 23)。1 桁の場合は前に空白文字が付加される
%l	12 時間表示で表した時 (1 から 12)。1 桁の場合は前に空白文字が付加される
%m	月 (1 から 12)。1 桁の場合は前に 0 が付加される
%M	分 (00 から 59)。先行する 0 は任意
%n	復帰改行を挿入
%p	午前または午後を表す値 (ロケール固有)
%r	%p を伴った、12 時間表示形式で表した時刻
%R	%H:%M 形式で表した時刻
%S	秒 (00 から 61)。値の範囲が (00 から 59) ではなく (00 から 61) であるのは、うるう秒および二重うるう秒に対処するため
%t	タブを挿入
%T	%H:%M:%S 形式で表した時刻
%u	曜日を表す番号 (1 から 7)。日曜日が 1 (「注意事項」を参照)
%U	その年の何週目かを表す数値 (00 から 53)。第 1 週はその年の最初の日曜日 から始まるとする
%V	ISO 8601 で何週目かを表す数値 (01 から 53)。ISO 8601 の週をベースとするシステムでは、週は月曜日から始まり、1 月 4 日およびその年の最初の木曜日の両方を含む週が第 1 週となる。その年の最初の月曜日が 1 月 1 日、2 日、3 日、または 4 日の場合には、その週は前年の最終週の一部になる (「注意事項」を参照)
%w	曜日を表す番号 (0 から 6)。日曜日が 0
%W	その年の何週目かを表す数値 (00 から 53)。第 1 週はその年の最初の月曜日 から始まるとする
%x	適切な日付の表現 (ロケール固有)
%X	適切な時間の表現 (ロケール固有)
%y	西暦年の下 2 桁 (00 から 99)
%Y	西暦年の完全表示 (たとえば 1993)
%Z	タイムゾーン名 (タイムゾーンがない場合には文字なし)

上記の変換仕様または後述する変更指定のどれにも該当しない指定を記述すると、変換結果は未定義で 0 が返されます。

%U と %W (このほか、後述の %OU と %OW) とでは、何曜日を週の第 1 日目とするかが異なっています。たとえば、1 月の第 1 週目 (週番号 1) は、%U では日曜日から、%W では月曜日からはじめます。また、週番号 0 には、%U では 1 月の第 1 日曜日より前の日が、%W では 1 月の第 1 月曜日より前の日が含まれています。

#### 変換仕様の変更

上記の変換仕様に、変更を表す文字 E または O を付加することができます。このようにすると、代替形式または代替指定で値を得ることが可能です。希望した代替形式または代替指定が 現ロケール中に存在していなければ、変更を示す文字を指定しなかった場合の形式で値が得られます。

%Ec	適切な日付および時刻の代替形式 (ロケール固有)
%EC	代替表示形式として指定されている年号 (ロケール固有)
%Eg	代替表示形式として指定されている年号 (%EC) に対応した週をベースとした年 (ロケール固有)
%EG	完全形式の週をベースとした代替年表示
%Ex	日付の代替表示形式 (ロケール固有)
%EX	時刻の代替表示形式 (ロケール固有)
%Ey	代替表示形式として指定されている年号 (%EC) に対応した年 (ロケール固有)
%EY	完全形式の代替年表示
%Od	日をロケール固有の代替数値記号で表す
%Oe	%Od と同じ
%Og	ロケール固有の週をベースとした代替表示の年 (%c に対応) の値を、ロケール固有の代替数値記号で表す
%OH	24 時間表示での時をロケール固有の代替数値記号で表す
%OI	12 時間表示での時をロケール固有の代替数値記号で表す
%Om	月をロケール固有の代替数値記号で表す
%OM	分をロケール固有の代替数値記号で表す
%OS	秒をロケール固有の代替数値記号で表す
%Ou	週日を示す値をロケール固有の代替数値記号にある数字で表す
%OU	その年の何週目かをロケール固有の代替数値記号で表す。週は日曜日からはじまるとする
%Ow	曜日を指示する値をロケール固有の代替数値記号で表す。日曜日を 0 とする
%OW	その年の何週目かをロケール固有の代替数値記号で表す。週は月曜日からはじまるとする

## cftime(3C)

%Oy	ロケール固有の代替表示の年 (%c に対応) の値を、ロケール固有の代替数値記号で表す						
出力言語の選択	デフォルトでは、 <code>strftime()</code> 、 <code>cftime()</code> 、 <code>ascftime()</code> は米国英語で表示されます。 <code>setlocale()</code> で <code>LC_TIME</code> カテゴリのロケールを設定すると、 <code>strftime()</code> 、 <code>cftime()</code> 、 <code>ascftime()</code> の実行結果を特定の言語で出力できるようになります。						
タイムゾーン	<code>tzset(3C)</code> を呼び出した場合と同じタイムゾーン情報を使用します。						
戻り値	<code>strftime()</code> 、 <code>cftime()</code> 、 <code>ascftime()</code> 関数は、 <code>s</code> が示す配列に書き出した文字数を返します。これには終了 <code>NULL</code> 文字は含まれません。生成された文字の合計数 (終了 <code>NULL</code> 文字を含む) が <code>maxsize</code> を超えていると、 <code>strftime()</code> は、0 を返し、配列の内容は未確定になります。						
使用例	<p>例 1 An example of the <code>strftime()</code> function.</p> <p>ここでは、POSIX ロケールの <code>strftime()</code> の使用例を示します。<code>tmpr</code> が指す構造体に 1986 年 8 月 28 日木曜日 12 時 44 分 36 秒に対応する値が入っている場合、文字列 <code>str</code> の内容がどうなるかを示します。</p> <pre>strftime (str, strsize, "%A %b %d %j", tmpr)</pre> <p>これにより、<code>str</code> の内容は "Thursday Aug 28 240" になります。</p>						
属性	次の属性については <code>attributes(5)</code> のマニュアルページを参照してください。						
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">属性タイプ</th> <th style="text-align: center;">属性値</th> </tr> </thead> <tbody> <tr> <td>MT レベル</td> <td>MT-safe</td> </tr> <tr> <td>CSI</td> <td>対応済み</td> </tr> </tbody> </table>	属性タイプ	属性値	MT レベル	MT-safe	CSI	対応済み
属性タイプ	属性値						
MT レベル	MT-safe						
CSI	対応済み						
関連項目	<code>date(1)</code> , <code>ctime(3C)</code> , <code>mkttime(3C)</code> , <code>setlocale(3C)</code> , <code>strptime(3C)</code> , <code>tzset(3C)</code> , <code>TIMEZONE(4)</code> , <code>zoneinfo(4)</code> , <code>attributes(5)</code> , <code>environ(5)</code> , <code>standards(5)</code>						
注意事項	<p>Solaris 7 では、<code>%v</code> の変換仕様が変わりました。この変更は、現時点で公表されている ISO C9x 標準の決定に基づいています。以前の仕様では、1 月 1 日を含む週に、4 日未満の日数の新しい年が含まれている場合、その週は前年の第 53 週となりましたが、ISO C9x 標準委員会によって、この仕様が正しくないと判断されました。</p> <p><code>%g</code>、<code>%G</code>、<code>%Eg</code>、<code>%EG</code>、<code>%Og</code> の変換仕様が Solaris 7 に追加されました。この変更は、現時点で公表されている ISO C9x 標準の決定に基づいています。これらの仕様は ISO C9x 標準の決定によって変更される可能性があり、変更前の仕様との互換性が保たれないこともあります。</p> <p>Solaris 8 リリースでは、<code>%u</code> の変換仕様に変更されました。この変更は、XPG4 仕様に基づいたものです。</p>						

cftime(3C)

%z 指示子と zoneinfo タイムゾーンを使用している場合に、入力日時が 20:45:52 UTC, December 13, 1901 から 03:14:07 UTC, January 19, 2038 までの範囲外にある場合、タイムゾーン名が正しくないことがあります。

## cset(3C)

名前 cset, csetlen, csetcol, csetno, wcsetno – EUC コードセットに関する情報の取得

形式

```
#include <euc.h>

int csetlen(int codeset);

int csetcol(int codeset);

int csetno(unsigned char c);

#include <widec.h>

int wcsetno(wchar_t pc);
```

機能説明 csetlen() と csetcol() は、ともにコードセット番号 *codeset* を取得します。コードセット番号は 0、1、2、または 3 を指定してください。csetlen() 関数は、所定の拡張 UNIX コード (EUC) コードセットの文字 (コードセット 2 およびコードセット 3 用のシングルシフト文字 SS2 および SS3 を除く) を表現するのに必要なバイト数を返します。csetcol() 関数は、所定の EUC コードセットの文字がディスプレイ上で占めるカラム数を返します。

csetno() 関数は、最初のバイトが *c* である EUC 文字に対するコードセット番号 (0、1、2、または 3) を返すマクロです。たとえば、

```
#include<euc.h> . . . x+=csetcol(csetno(c));
```

によって、カウンタ *x* (カーソル位置など) は、最初のバイトが *c* の文字幅分増加します。

wcsetno() 関数は、所定のワイド文字 *pc* に対するコードセット番号 (0、1、2、または 3) を返すマクロとして定義されています。たとえば、

```
#include<euc.h> #include<widec.h> . . . x+=csetcol(wcsetno(pc));
```

によって、カウンタ *x* (カーソル位置など) は、ワイド文字 *pc* の幅だけ増加します。

使用法 cset(), csetlen(), csetcol(), csetno() または wcsetno() 関数は setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッドアプリケーション上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 setlocale(3C) euclen(3C), attributes(5)



名前 cset, csetlen, csetcol, csetno, wcsetno – EUC コードセットに関する情報の取得

形式

```
#include <euc.h>

int csetlen(int codeset);

int csetcol(int codeset);

int csetno(unsigned char c);

#include <wdec.h>

int wcsetno(wchar_t pc);
```

機能説明

csetlen() と csetcol() は、ともにコードセット番号 *codeset* を取得します。コードセット番号は 0、1、2、または 3 を指定してください。csetlen() 関数は、所定の拡張 UNIX コード (EUC) コードセットの文字 (コードセット 2 およびコードセット 3 用のシングルシフト文字 SS2 および SS3 を除く) を表現するのに必要なバイト数を返します。csetcol() 関数は、所定の EUC コードセットの文字がディスプレイ上で占めるカラム数を返します。

csetno() 関数は、最初のバイトが *c* である EUC 文字に対するコードセット番号 (0、1、2、または 3) を返すマクロです。たとえば、

```
#include<euc.h> . . . x+=csetcol(csetno(c));
```

によって、カウンタ *x* (カーソル位置など) は、最初のバイトが *c* の文字幅分増加します。

wcsetno() 関数は、所定のワイド文字 *pc* に対するコードセット番号 (0、1、2、または 3) を返すマクロとして定義されています。たとえば、

```
#include<euc.h> #include<wdec.h> . . . x+=csetcol(wcsetno(pc));
```

によって、カウンタ *x* (カーソル位置など) は、ワイド文字 *pc* の幅だけ増加します。

使用法

cset(), csetlen(), csetcol(), csetno() または wcsetno() 関数は setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッドアプリケーション上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 setlocale(3C) euclen(3C), attributes(5)

## csetlen(3C)

名前 cset, csetlen, csetcol, csetno, wcsetno – EUC コードセットに関する情報の取得

形式

```
#include <euc.h>

int csetlen(int codeset);

int csetcol(int codeset);

int csetno(unsigned char c);

#include <widec.h>

int wcsetno(wchar_t pc);
```

機能説明 csetlen() と csetcol() は、ともにコードセット番号 *codeset* を取得します。コードセット番号は 0、1、2、または 3 を指定してください。csetlen() 関数は、所定の拡張 UNIX コード (EUC) コードセットの文字 (コードセット 2 およびコードセット 3 用のシングルシフト文字 SS2 および SS3 を除く) を表現するのに必要なバイト数を返します。csetcol() 関数は、所定の EUC コードセットの文字がディスプレイ上で占めるカラム数を返します。

csetno() 関数は、最初のバイトが *c* である EUC 文字に対するコードセット番号 (0、1、2、または 3) を返すマクロです。たとえば、

```
#include<euc.h> . . . x+=csetcol(csetno(c));
```

によって、カウンタ *x* (カーソル位置など) は、最初のバイトが *c* の文字幅分増加します。

wcsetno() 関数は、所定のワイド文字 *pc* に対するコードセット番号 (0、1、2、または 3) を返すマクロとして定義されています。たとえば、

```
#include<euc.h> #include<widec.h> . . . x+=csetcol(wcsetno(pc));
```

によって、カウンタ *x* (カーソル位置など) は、ワイド文字 *pc* の幅だけ増加します。

使用法 cset(), csetlen(), csetcol(), csetno() または wcsetno() 関数は setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッドアプリケーション上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 setlocale(3C) euclen(3C), attributes(5)

名前 cset, csetlen, csetcol, csetno, wcsetno – EUC コードセットに関する情報の取得

形式 #include <euc.h>

```
int csetlen(int codeset);
```

```
int csetcol(int codeset);
```

```
int csetno(unsigned char c);
```

```
#include <widec.h>
```

```
int wcsetno(wchar_t pc);
```

機能説明 csetlen() と csetcol() は、ともにコードセット番号 *codeset* を取得します。コードセット番号は 0、1、2、または 3 を指定してください。csetlen() 関数は、所定の拡張 UNIX コード (EUC) コードセットの文字 (コードセット 2 およびコードセット 3 用のシングルシフト文字 SS2 および SS3 を除く) を表現するのに必要なバイト数を返します。csetcol() 関数は、所定の EUC コードセットの文字がディスプレイ上で占めるカラム数を返します。

csetno() 関数は、最初のバイトが *c* である EUC 文字に対するコードセット番号 (0、1、2、または 3) を返すマクロです。たとえば、

```
#include<euc.h> . . . x+=csetcol(csetno(c));
```

によって、カウンタ *x* (カーソル位置など) は、最初のバイトが *c* の文字幅分増加します。

wcsetno() 関数は、所定のワイド文字 *pc* に対するコードセット番号 (0、1、2、または 3) を返すマクロとして定義されています。たとえば、

```
#include<euc.h> #include<widec.h> . . . x+=csetcol(wcsetno(pc));
```

によって、カウンタ *x* (カーソル位置など) は、ワイド文字 *pc* の幅だけ増加します。

使用法 cset(), csetlen(), csetcol(), csetno() または wcsetno() 関数は setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッドアプリケーション上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 setlocale(3C) euclen(3C), attributes(5)

## ctype(3C)

名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に __XPG4_CHAR_CLASS__ を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任

		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字のASCII文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字のASCII文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の10進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の16進数の文字([0-9]、[A-F]、または[a-f])について検査します。
標準準拠	<code>isxdigit()</code>	任意の16進数の文字([0-9]、[A-F]、または[a-f])、あるいは16進数を表現する(10から15までの数字も含む)現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字(文字または数字)について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィールド(標準空白文字)について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## ctype(3C)

`iscntrl()` 文字集合に定義されている「制御文字」について検査します。

`isascii()` 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 `isdigit()`、`isxdigit()`、`islower()`、`isupper()`、`isalpha()`、`isalnum()`、`isspace()`、`iscntrl()`、`ispunct()`、`isprint()`、`isgraph()`、または `isascii()` マクロは、`setlocale(3C)` がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 `setlocale(3C)`、`stdio(3C)`、`ascii(5)`、`environ(5)`、`standards(5)`

名前	gettext, dgettext, dcgettext, ngettext, dngettext, dcngettext, textdomain, bindtextdomain, bind_textdomain_codeset – メッセージ処理関数
形式	
Solaris および GNU 互換	<pre>#include &lt;libintl.h&gt; char *gettext(const char *msgid); char *dgettext(const char *domainname, const char *msgid); char *textdomain(const char *domainname); char *bindtextdomain(const char *domainname, const char *dirname); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcgettext(const char *domainname, const char *msgid, int category);</pre>
GNU 互換	<pre>#include &lt;libintl.h&gt; char *ngettext(const char *msgid1, const char *msgid2, unsigned long int n); char *dngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n); char *bind_textdomain_codeset(const char *domainname, const char *codeset); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n, int category);</pre>
機能説明	<p>gettext(), dgettext(), dcgettext() 関数は、特定のドメイン内において、また、現在のロケールにおいて、指定の <i>msgid</i> 引数に基づいたターゲット文字列を取得します。gettext(), dgettext(), dcgettext() から返される文字列の長さは、これらの関数が呼び出されるまでは未定です。<i>msgid</i> 引数は、NULL で終わる文字列です。</p> <p>ngettext(), dngettext(), および dcngettext() 関数はそれぞれ、gettext(), dgettext(), および dcgettext() と同等ですが、GNU 互換メッセージカタログだけを処理することができ、複数形の処理方法が異なります。メッセージ文字列を検索するとき、ngettext(), dngettext(), および dcngettext() 関数は <i>msgid1</i> 引数を検索鍵として使用し、<i>n</i> 引数で複数形を決定します。メッセージカタログが見つからない場合、<i>n</i> == 1 であれば <i>msgid1</i> が返され、そうでなければ <i>msgid2</i> が返されます。</p> <p>環境変数 NLSPATH (environ(5) を参照) が、最初に LC_MESSAGES カタログの位置として検索されます。現在のロケールの LC_MESSAGES カテゴリの設定によって、gettext() および dgettext() が文字列取得に使用するロケールが決まります。</p>

## dcgettext(3C)

*category* 引数は `dcgettext()` の使用するロケールを決めます。NLSPATH が定義されておらず、現在のロケールが C ロケールの場合、`gettext()`、`gettext()`、`dcgettext()` は、渡されたメッセージ文字列をそのまま返します。C 以外のロケールでは、NLSPATH が定義されていない場合、あるいは、NLSPATH で指定したコンポーネントでメッセージカタログが見つからない場合、メッセージカタログの検索には、次の段落で説明するスキームが使用されます。

LANGUAGE 環境変数は、使用する GNU 互換メッセージカタログを決定するために調べられます。LANGUAGE の値はコロン (:) で区切られたロケール名のリストです。LANGUAGE が定義されている場合、リストに指定されている順番で各ロケールが検索されます。そして、GNU 互換メッセージカタログが見つかった場合、そのメッセージが返されます。GNU 互換メッセージが見つかったが、対応する *msgid* が見つからない場合、*msgid* 文字列が返されます。LANGUAGE が定義されていない場合、あるいは、LANGUAGE の処理中に Solaris メッセージカタログが見つかったが GNU 互換メッセージカタログが見つからない場合、*dirname/locale/category/domainname.mo* というパス名でメッセージカタログが検索されます。ここで、*dirname* は `bindtextdomain()` で指定されたディレクトリ、*locale* はロケール名、*category* は LC\_MESSAGES または LC\_XXX のどちらかです。LC\_MESSAGES は `gettext()`、`dgettext()`、`ngettext()`、または `dngettext()` を呼び出した場合に使用されます。LC\_XXX は `dcgettext()` または `dcngettext()` を呼び出した場合に使用され、XXX 部分は *category* 引数で指定したロケールカテゴリ名と同じになります。

`gettext()` と `ngettext()` の場合、使用されるドメインは、最後に行なった `textdomain()` の有効な呼び出しによって設定されます。`textdomain()` の有効な呼び出しが行われていない場合は、デフォルト時のドメイン (`messages`) が使用されます。

`dgettext()`、`dngettext()`、`dcngettext()`、および `dcgettext()` の場合、使用されるドメインは *domainname* 引数で指定されます。この *domainname* 引数は、構文と意味において、`textdomain()` の *domainname* 引数と同等ですが、このドメインの選択が `dgettext()`、`dcgettext()`、`dngettext()` および `dcngettext()` 関数の呼び出し中のみ有効であるという点で異なります。

`textdomain()` 関数は、アクティブな LC\_MESSAGES ロケールカテゴリの現在のドメイン名の設定または照会を行います。*domainname* 引数は、正当なファイル名に使用できる文字のみからなる、NULL で終わる文字列です。

*domainname* 引数は、システム内の、一意のドメイン名です。1つのシステム内に、同一ドメインのバージョンが複数存在する場合は、`bindtextdomain()` を使用して名前空間の衝突を回避することができます。`textdomain()` を呼び出さない場合には、デフォルト時のドメインが選択されます。最後に行なった `textdomain()` の有効な呼び出しによるドメインの設定は、このあとに行う `setlocale(3C)` および `gettext()` の呼び出しにおいても有効です。

*domainname* 引数は、現在アクティブな、LC\_MESSAGES ロケールに適用されます。



*domainname* に NULL ポインタを設定して `textdomain()` を呼び出すことによって、ドメインの現在の状態に影響することなく、ドメインの現在の設定を照会することができます。*domainname* 引数に NULL 文字列を設定して `textdomain()` を呼び出すと、ドメインの設定は、デフォルト時のドメイン (`messages`) になります。

`bindtextdomain()` 関数は、メッセージドメインのパス部分 *domainname* と、*dirname* に入っている値を結合します。*domainname* が空文字列ではなく、まだ結合されていない場合には、`bindtextdomain()` は、*domainname* と *dirname* を結合します。

*domainname* が空文字列ではなく、すでに結合されている場合には、`bindtextdomain()` は、事前に行われていた結合部分を *dirname* に置き換えます。*dirname* 引数に使用できるのは、`gettext()`、`dgettext()`、または `dcgettext()` が呼び出されたときに解決される、絶対パス名または相対パス名です。*domainname* が NULL ポインタまたは空文字列の場合は、`bindtextdomain()` は NULL を返します。ユーザー定義のドメイン名は、`SYS_` で始めることはできません。この文字列で始まるドメイン名は、システムで使用するために予約されています。

`bind_textdomain_codeset()` 関数を使用すると、ドメイン *domainname* のメッセージカタログ用の出力コードセットを指定できます。*codeset* 引数は、`iconv_open(3C)` 関数に対して使用可能な有効なコードセット名であるか、空ポインタである必要があります。*codeset* 引数が空ポインタの場合、`bind_textdomain_codeset()` はドメイン *domainname* 用に現在選択されているコードセットを返します。コードセットが選択されていない場合、`bind_textdomain_codeset()` は空ポインタを返します。`bind_textdomain_codeset()` 関数は複数回使用できます。同じ *domainname* 引数を指定して `bind_textdomain_codeset()` 関数を複数回使用した場合、先行する呼び出しで行われた設定は後続の呼び出しによって無効にされます。`bind_textdomain_codeset()` 関数は、選択されたコードセット名を含む文字列へのポインタを返します。この文字列は `bind_textdomain_codeset()` 関数に内部的に割り当てられているため、ユーザーが変更してはなりません。

戻り値

`gettext()`、`dgettext()`、および `dcgettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*msgid* 文字列を返します。

`ngettext()`、`dngettext()`、および `dcngettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*n* == 1 であれば、*msgid1* を返します。そうでなければ、*msgid2* を返します。

`gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` が返す文字列の個々のバイトには、空文字以外のあらゆる値が入ります。*msgid* が NULL ポインタである場合の戻り値は未定義です。返される文字列は、プログラムで修正してはならず、この後で `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` または `setlocale(3C)` を呼び出すことにより、無効になることがあります。`dgettext()`、`dcgettext()`、`dngettext()`、または `dcngettext()` に対する *domainname* 引数が NULL ポインタである場合、現在 `textdomain()` によってバインドされているドメインが使用されます。

## dcgettext(3C)

`textdomain()` の正常な戻り値は、ドメインの現在の設定が入っている文字列へのポインタです。`domainname` が NULL ポインタの場合は、`textdomain()` は現在のドメインが入っている文字列へのポインタを返します。`textdomain()` を先に呼び出していない場合に `domainname` を空文字列にすると、デフォルト時のドメイン名が返されます。デフォルト時のドメイン名は、`messages` です。`textdomain()` が失敗した場合、NULL ポインタが返されます。

`bindtextdomain()` からの戻り値は、`dirname` が入った NULL で終わる文字列です。`dirname` が NULL の場合は、`domainname` と結び付けられたディレクトリバインディングです。何も結合されていない場合には、デフォルトの戻り値は `/usr/lib/locale` になります。`domainname` が NULL ポインタまたは空文字列の場合、`bindtextdomain()` は何も行わず、NULL ポインタを返します。返された文字列を呼び出し側で変更してはなりません。`bindtextdomain()` が失敗した場合、NULL ポインタが返されます。

使用法 これらのルーチンでは、メッセージの長さに制限はありません。ただし、テキスト `domainname` は、TEXTDOMAINMAX (256) バイト以内に制限されています。

ロケール変更のために `setlocale(3C)` が呼び出されていない限り、マルチスレッドアプリケーションにおいて `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()`、`textdomain()`、および `bindtextdomain()` 関数を安全に使うことができます。

`gettext()`、`dgettext()`、`dcgettext()`、`textdomain()`、および `bindtextdomain()` 関数は Solaris メッセージカタログと GNU 互換メッセージカタログの両方を処理できます。`ngettext()`、`dngettext()`、`dcngettext()`、および `bind_textdomain_codeset()` 関数は GNU 互換メッセージカタログだけを処理できます。Solaris メッセージカタログと GNU 互換メッセージカタログの詳細については、`msgfmt(1)` のマニュアルページを参照してください。

## ファイル

`/usr/lib/locale`

メッセージ・ドメイン・ファイルのデフォルトパス部分

`/usr/lib/locale/locale/LC_MESSAGES/domainname.mo`

言語 `locale` および `domainname` 用メッセージの入っているファイルのシステムデフォルト位置

`/usr/lib/locale/locale/LC_XXX/domainname.mo`

`LC_XXX` が `LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` である場合に、`dcgettext()` 呼び出し用の言語 `locale` および `domainname` メッセージの入っているファイルのシステムデフォルト位置

`dirname/locale/LC_MESSAGES/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の、ドメイン `domainname` およびパス部分 `dirname` 用メッセージが入っているファイルの位置

`dirname/locale/LC_XXX/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の `LC_XXX` が、`LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` のいずれかである場合 `dcgettext()` 呼び出し用のドメイン `domainname`、言語

dcgettext(3C)

*locale*、およびパス部分 *dirname* メッセージが入っているファイルの位置

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 `msgfmt(1)`, `xgettext(1)`, `iconv_open(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

## dcgettext(3C)

名前	gettext, dgettext, dcgettext, ngettext, dngettext, dcgettext, textdomain, bindtextdomain, bind_textdomain_codeset – メッセージ処理関数
形式	
Solaris および GNU 互換	<pre>#include &lt;libintl.h&gt; char *gettext(const char *msgid); char *dgettext(const char *domainname, const char *msgid); char *textdomain(const char *domainname); char *bindtextdomain(const char *domainname, const char *dirname); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcgettext(const char *domainname, const char *msgid, int category);</pre>
GNU 互換	<pre>#include &lt;libintl.h&gt; char *ngettext(const char *msgid1, const char *msgid2, unsigned long int n); char *dngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n); char *bind_textdomain_codeset(const char *domainname, const char *codeset); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n, int category);</pre>
機能説明	<p>gettext(), dgettext(), dcgettext() 関数は、特定のドメイン内において、また、現在のロケールにおいて、指定の <i>msgid</i> 引数に基づいたターゲット文字列を取得します。gettext(), dgettext(), dcgettext() から返される文字列の長さは、これらの関数が呼び出されるまでは未定です。<i>msgid</i> 引数は、NULL で終わる文字列です。</p> <p>ngettext(), dngettext(), および dcngettext() 関数はそれぞれ、gettext(), dgettext(), および dcgettext() と同等ですが、GNU 互換メッセージカタログだけを処理することができ、複数形の処理方法が異なります。メッセージ文字列を検索するとき、ngettext(), dngettext(), および dcngettext() 関数は <i>msgid1</i> 引数を検索鍵として使用し、<i>n</i> 引数で複数形を決定します。メッセージカタログが見つからない場合、<i>n</i> == 1 であれば <i>msgid1</i> が返され、そうでなければ <i>msgid2</i> が返されます。</p> <p>環境変数 NLSPATH (environ(5) を参照) が、最初に LC_MESSAGES カタログの位置として検索されます。現在のロケールの LC_MESSAGES カテゴリの設定によって、gettext() および dgettext() が文字列取得に使用するロケールが決まります。</p>

*category* 引数は `dcgettext()` の使用するロケールを決めます。NLSPATH が定義されておらず、現在のロケールが C ロケールの場合、`gettext()`、`gettext()`、`dcgettext()` は、渡されたメッセージ文字列をそのまま返します。C 以外のロケールでは、NLSPATH が定義されていない場合、あるいは、NLSPATH で指定したコンポーネントでメッセージカタログが見つからない場合、メッセージカタログの検索には、次の段落で説明するスキームが使用されます。

LANGUAGE 環境変数は、使用する GNU 互換メッセージカタログを決定するために調べられます。LANGUAGE の値はコロン (:) で区切られたロケール名のリストです。LANGUAGE が定義されている場合、リストに指定されている順番で各ロケールが検索されます。そして、GNU 互換メッセージカタログが見つかった場合、そのメッセージが返されます。GNU 互換メッセージが見つかったが、対応する *msgid* が見つからない場合、*msgid* 文字列が返されます。LANGUAGE が定義されていない場合、あるいは、LANGUAGE の処理中に Solaris メッセージカタログが見つかったが GNU 互換メッセージカタログが見つからない場合、*dirname/locale/category/domainname.mo* というパス名でメッセージカタログが検索されます。ここで、*dirname* は `bindtextdomain()` で指定されたディレクトリ、*locale* はロケール名、*category* は LC\_MESSAGES または LC\_XXX のどちらかです。LC\_MESSAGES は `gettext()`、`dgettext()`、`ngettext()`、または `dngettext()` を呼び出した場合に使用されます。LC\_XXX は `dcgettext()` または `dcngettext()` を呼び出した場合に使用され、XXX 部分は *category* 引数で指定したロケールカテゴリ名と同じになります。

`gettext()` と `ngettext()` の場合、使用されるドメインは、最後に行なった `textdomain()` の有効な呼び出しによって設定されます。`textdomain()` の有効な呼び出しが行われていない場合は、デフォルト時のドメイン (`messages`) が使用されます。

`dgettext()`、`dngettext()`、`dcngettext()`、および `dcgettext()` の場合、使用されるドメインは *domainname* 引数で指定されます。この *domainname* 引数は、構文と意味において、`textdomain()` の *domainname* 引数と同等ですが、このドメインの選択が `dgettext()`、`dcgettext()`、`dngettext()` および `dcngettext()` 関数の呼び出し中にのみ有効であるという点で異なります。

`textdomain()` 関数は、アクティブな LC\_MESSAGES ロケールカテゴリの現在のドメイン名の設定 または照会を行います。*domainname* 引数は、正当なファイル名に使用できる文字のみからなる、NULL で終わる文字列です。

*domainname* 引数は、システム内の、一意のドメイン名です。1つのシステム内に、同一ドメインのバージョンが複数存在する場合は、`bindtextdomain()` を使用して名前空間の衝突を回避することができます。`textdomain()` を呼び出さない場合には、デフォルト時のドメインが選択されます。最後に行なった `textdomain()` の有効な呼び出しによるドメインの設定は、このあとに行う `setlocale(3C)` および `gettext()` の呼び出しにおいても有効です。

*domainname* 引数は、現在アクティブな、LC\_MESSAGES ロケールに適用されます。

## dcngettext(3C)

*domainname* に NULL ポインタを設定して `textdomain()` を呼び出すことによって、ドメインの現在の状態に影響することなく、ドメインの現在の設定を照会することができます。*domainname* 引数に NULL 文字列を設定して `textdomain()` を呼び出すと、ドメインの設定は、デフォルト時のドメイン (`messages`) になります。

`bindtextdomain()` 関数は、メッセージドメインのパス部分 *domainname* と、*dirname* に入っている値を結合します。*domainname* が空文字列ではなく、まだ結合されていない場合には、`bindtextdomain()` は、*domainname* と *dirname* を結合します。

*domainname* が空文字列ではなく、すでに結合されている場合には、`bindtextdomain()` は、事前に行われていた結合部分を *dirname* に置き換えます。*dirname* 引数に使用できるのは、`gettext()`、`dgettext()`、または `dcgettext()` が呼び出されたときに解決される、絶対パス名または相対パス名です。*domainname* が NULL ポインタまたは空文字列の場合は、`bindtextdomain()` は NULL を返します。ユーザー定義のドメイン名は、`sys_` で始めることはできません。この文字列で始まるドメイン名は、システムで使用するために予約されています。

`bind_textdomain_codeset()` 関数を使用すると、ドメイン *domainname* のメッセージカタログ用の出力コードセットを指定できます。*codeset* 引数は、`iconv_open(3C)` 関数に対して使用可能な有効なコードセット名であるか、空ポインタである必要があります。*codeset* 引数が空ポインタの場合、`bind_textdomain_codeset()` はドメイン *domainname* 用に現在選択されているコードセットを返します。コードセットが選択されていない場合、`bind_textdomain_codeset()` は空ポインタを返します。`bind_textdomain_codeset()` 関数は複数回使用できます。同じ *domainname* 引数を指定して `bind_textdomain_codeset()` 関数を複数回使用した場合、先行する呼び出しで行われた設定は後続の呼び出しによって無効にされます。`bind_textdomain_codeset()` 関数は、選択されたコードセット名を含む文字列へのポインタを返します。この文字列は `bind_textdomain_codeset()` 関数に内部的に割り当てられているため、ユーザーが変更してはなりません。

戻り値

`gettext()`、`dgettext()`、および `dcgettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*msgid* 文字列を返します。

`ngettext()`、`dngettext()`、および `dcngettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*n* == 1 であれば、*msgid1* を返します。そうでなければ、*msgid2* を返します。

`gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` が返す文字列の個々のバイトには、空文字以外のあらゆる値が入ります。*msgid* が NULL ポインタである場合の戻り値は未定義です。返される文字列は、プログラムで修正してはならず、この後で `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` または `setlocale(3C)` を呼び出すことにより、無効になることがあります。`dgettext()`、`dcgettext()`、`dngettext()`、または `dcngettext()` に対する *domainname* 引数が NULL ポインタである場合、現在 `textdomain()` によってバインドされているドメインが使用されます。

## dcngettext(3C)

`textdomain()` の正常な戻り値は、ドメインの現在の設定が入っている文字列へのポインタです。`domainname` が NULL ポインタの場合は、`textdomain()` は現在のドメインが入っている文字列へのポインタを返します。`textdomain()` を先に呼び出していない場合に `domainname` を空文字列にすると、デフォルト時のドメイン名が返されます。デフォルト時のドメイン名は、`messages` です。`textdomain()` が失敗した場合、NULL ポインタが返されます。

`bindtextdomain()` からの戻り値は、`dirname` が入った NULL で終わる文字列です。`dirname` が NULL の場合は、`domainname` と結び付けられたディレクトリバインディングです。何も結合されていない場合には、デフォルトの戻り値は `/usr/lib/locale` になります。`domainname` が NULL ポインタまたは空文字列の場合、`bindtextdomain()` は何も行わず、NULL ポインタを返します。返された文字列を呼び出し側で変更してはなりません。`bindtextdomain()` が失敗した場合、NULL ポインタが返されます。

使用法 これらのルーチンでは、メッセージの長さに制限はありません。ただし、テキスト `domainname` は、TEXTDOMAINMAX (256) バイト以内に制限されています。

ロケール変更のために `setlocale(3C)` が呼び出されていない限り、マルチスレッドアプリケーションにおいて `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()`、`textdomain()`、および `bindtextdomain()` 関数を安全に使うことができます。

`gettext()`、`dgettext()`、`dcgettext()`、`textdomain()`、および `bindtextdomain()` 関数は Solaris メッセージカタログと GNU 互換メッセージカタログの両方を処理できます。`ngettext()`、`dngettext()`、`dcngettext()`、および `bind_textdomain_codeset()` 関数は GNU 互換メッセージカタログだけを処理できます。Solaris メッセージカタログと GNU 互換メッセージカタログの詳細については、`msgfmt(1)` のマニュアルページを参照してください。

ファイル

`/usr/lib/locale`

メッセージ・ドメイン・ファイルのデフォルトパス部分

`/usr/lib/locale/locale/LC_MESSAGES/domainname.mo`

言語 `locale` および `domainname` 用メッセージの入っているファイルのシステムデフォルト位置

`/usr/lib/locale/locale/LC_XXX/domainname.mo`

`LC_XXX` が `LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、

`LC_MONETARY`、または `LC_MESSAGES` である場合に、`dcgettext()` 呼び出し用の言語 `locale` および `domainname` メッセージの入っているファイルのシステムデフォルト位置

`dirname/locale/LC_MESSAGES/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の、ドメイン `domainname` およびパス部分 `dirname` 用メッセージが入っているファイルの位置

`dirname/locale/LC_XXX/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の `LC_XXX` が、`LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` のいずれかである場合 `dcgettext()` 呼び出し用のドメイン `domainname`、言語

## dcngettext(3C)

属性 *locale*、およびパス部分 *dirname* メッセージが入っているファイルの位置  
次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 `msgfmt(1)`, `xgettext(1)`, `iconv_open(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`



名前 gettext, dgettext, dcgettext, ngettext, dngettext, dcngettext, textdomain, bindtextdomain, bind\_textdomain\_codeset – メッセージ処理関数

形式

Solaris および  
GNU 互換

```
#include <libintl.h>
char *gettext(const char *msgid);
char *dgettext(const char *domainname, const char *msgid);
char *textdomain(const char *domainname);
char *bindtextdomain(const char *domainname, const char *dirname);
```

GNU 互換

```
#include <libintl.h>
#include <locale.h>
char *dcgettext(const char *domainname, const char *msgid, int
category);
#include <libintl.h>
char *ngettext(const char *msgid1, const char *msgid2, unsigned long
int n);
char *dngettext(const char *domainname, const char *msgid1, const
char *msgid2, unsigned long int n);
char *bind_textdomain_codeset(const char *domainname, const char
*codeset);
#include <libintl.h>
#include <locale.h>
char *dcngettext(const char *domainname, const char *msgid1, const
char *msgid2, unsigned long int n, int category);
```

機能説明

gettext(), dgettext(), dcgettext() 関数は、特定のドメイン内において、また、現在のロケールにおいて、指定の *msgid* 引数に基づいたターゲット文字列を取得します。gettext(), dgettext(), dcgettext() から返される文字列の長さは、これらの関数が呼び出されるまでは未定です。*msgid* 引数は、NULL で終わる文字列です。

ngettext(), dngettext(), および dcngettext() 関数はそれぞれ、gettext(), dgettext(), および dcgettext() と同等ですが、GNU 互換メッセージカタログだけを処理することができ、複数形の処理方法が異なります。メッセージ文字列を検索するとき、ngettext(), dngettext(), および dcngettext() 関数は *msgid1* 引数を検索鍵として使用し、*n* 引数で複数形を決定します。メッセージカタログが見つからない場合、*n* == 1 であれば *msgid1* が返され、そうでなければ *msgid2* が返されます。

環境変数 NLSPATH (environ(5) を参照) が、最初に LC\_MESSAGES カタログの位置として検索されます。現在のロケールの LC\_MESSAGES カテゴリの設定によって、gettext() および dgettext() が文字列取得に使用するロケールが決まります。

## dgettext(3C)

*category* 引数は `dcgettext()` の使用するロケールを決めます。NLSPATH が定義されておらず、現在のロケールが C ロケールの場合、`gettext()`、`gettext()`、`dcgettext()` は、渡されたメッセージ文字列をそのまま返します。C 以外のロケールでは、NLSPATH が定義されていない場合、あるいは、NLSPATH で指定したコンポーネントでメッセージカタログが見つからない場合、メッセージカタログの検索には、次の段落で説明するスキームが使用されます。

LANGUAGE 環境変数は、使用する GNU 互換メッセージカタログを決定するために調べられます。LANGUAGE の値はコロン (:) で区切られたロケール名のリストです。LANGUAGE が定義されている場合、リストに指定されている順番で各ロケールが検索されます。そして、GNU 互換メッセージカタログが見つかった場合、そのメッセージが返されます。GNU 互換メッセージが見つかったが、対応する *msgid* が見つからない場合、*msgid* 文字列が返されます。LANGUAGE が定義されていない場合、あるいは、LANGUAGE の処理中に Solaris メッセージカタログが見つかったが GNU 互換メッセージカタログが見つからない場合、*dirname/locale/category/domainname.mo* というパス名でメッセージカタログが検索されます。ここで、*dirname* は `bindtextdomain()` で指定されたディレクトリ、*locale* はロケール名、*category* は LC\_MESSAGES または LC\_XXX のどちらかです。LC\_MESSAGES は `gettext()`、`dgettext()`、`ngettext()`、または `dngettext()` を呼び出した場合に使用されます。LC\_XXX は `dcgettext()` または `dcngettext()` を呼び出した場合に使用され、XXX 部分は *category* 引数で指定したロケールカテゴリ名と同じになります。

`gettext()` と `ngettext()` の場合、使用されるドメインは、最後に行なった `textdomain()` の有効な呼び出しによって設定されます。`textdomain()` の有効な呼び出しが行われていない場合は、デフォルト時のドメイン (`messages`) が使用されます。

`dgettext()`、`dngettext()`、`dcngettext()`、および `dcgettext()` の場合、使用されるドメインは *domainname* 引数で指定されます。この *domainname* 引数は、構文と意味において、`textdomain()` の *domainname* 引数と同等ですが、このドメインの選択が `dgettext()`、`dcgettext()`、`dngettext()` および `dcngettext()` 関数の呼び出し中のみ有効であるという点で異なります。

`textdomain()` 関数は、アクティブな LC\_MESSAGES ロケールカテゴリの現在のドメイン名の設定または照会を行います。*domainname* 引数は、正当なファイル名に使用できる文字のみからなる、NULL で終わる文字列です。

*domainname* 引数は、システム内の、一意のドメイン名です。1つのシステム内に、同一ドメインのバージョンが複数存在する場合は、`bindtextdomain()` を使用して名前空間の衝突を回避することができます。`textdomain()` を呼び出さない場合には、デフォルト時のドメインが選択されます。最後に行なった `textdomain()` の有効な呼び出しによるドメインの設定は、このあとに行う `setlocale(3C)` および `gettext()` の呼び出しにおいても有効です。

*domainname* 引数は、現在アクティブな、LC\_MESSAGES ロケールに適用されます。

## dgettext(3C)

*domainname* に NULL ポインタを設定して `textdomain()` を呼び出すことによって、ドメインの現在の状態に影響することなく、ドメインの現在の設定を照会することができます。*domainname* 引数に NULL 文字列を設定して `textdomain()` を呼び出すと、ドメインの設定は、デフォルト時のドメイン (`messages`) になります。

`bindtextdomain()` 関数は、メッセージドメインのパス部分 *domainname* と、*dirname* に入っている値を結合します。*domainname* が空文字列ではなく、まだ結合されていない場合には、`bindtextdomain()` は、*domainname* と *dirname* を結合します。

*domainname* が空文字列ではなく、すでに結合されている場合には、`bindtextdomain()` は、事前に行われていた結合部分を *dirname* に置き換えます。*dirname* 引数に使用できるのは、`gettext()`、`dgettext()`、または `dcgettext()` が呼び出されたときに解決される、絶対パス名または相対パス名です。*domainname* が NULL ポインタまたは空文字列の場合は、`bindtextdomain()` は NULL を返します。ユーザー定義のドメイン名は、`SYS_` で始めることはできません。この文字列で始まるドメイン名は、システムで使用するために予約されています。

`bind_textdomain_codeset()` 関数を使用すると、ドメイン *domainname* のメッセージカタログ用の出力コードセットを指定できます。*codeset* 引数は、`iconv_open(3C)` 関数に対して使用可能な有効なコードセット名であるか、空ポインタである必要があります。*codeset* 引数が空ポインタの場合、`bind_textdomain_codeset()` はドメイン *domainname* 用に現在選択されているコードセットを返します。コードセットが選択されていない場合、`bind_textdomain_codeset()` は空ポインタを返します。`bind_textdomain_codeset()` 関数は複数回使用できます。同じ *domainname* 引数を指定して `bind_textdomain_codeset()` 関数を複数回使用した場合、先行する呼び出しで行われた設定は後続の呼び出しによって無効にされます。`bind_textdomain_codeset()` 関数は、選択されたコードセット名を含む文字列へのポインタを返します。この文字列は `bind_textdomain_codeset()` 関数に内部的に割り当てられているため、ユーザーが変更してはなりません。

### 戻り値

`gettext()`、`dgettext()`、および `dcgettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*msgid* 文字列を返します。

`ngettext()`、`dngettext()`、および `dcngettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*n* == 1 であれば、*msgid1* を返します。そうでなければ、*msgid2* を返します。

`gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` が返す文字列の個々のバイトには、空文字以外のあらゆる値が入ります。*msgid* が NULL ポインタである場合の戻り値は未定義です。返される文字列は、プログラムで修正してはならず、この後で `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` または `setlocale(3C)` を呼び出すことにより、無効になることがあります。`dgettext()`、`dcgettext()`、`dngettext()`、または `dcngettext()` に対する *domainname* 引数が NULL ポインタである場合、現在 `textdomain()` によってバインドされているドメインが使用されます。

## dgettext(3C)

`textdomain()` の正常な戻り値は、ドメインの現在の設定が入っている文字列へのポインタです。`domainname` が NULL ポインタの場合は、`textdomain()` は現在のドメインが入っている文字列へのポインタを返します。`textdomain()` を先に呼び出していない場合に `domainname` を空文字列にすると、デフォルト時のドメイン名が返されます。デフォルト時のドメイン名は、`messages` です。`textdomain()` が失敗した場合、NULL ポインタが返されます。

`bindtextdomain()` からの戻り値は、`dirname` が入った NULL で終わる文字列です。`dirname` が NULL の場合は、`domainname` と結び付けられたディレクトリバインディングです。何も結合されていない場合には、デフォルトの戻り値は `/usr/lib/locale` になります。`domainname` が NULL ポインタまたは空文字列の場合、`bindtextdomain()` は何も行わず、NULL ポインタを返します。返された文字列を呼び出し側で変更してはなりません。`bindtextdomain()` が失敗した場合、NULL ポインタが返されます。

使用法 これらのルーチンでは、メッセージの長さに制限はありません。ただし、テキスト `domainname` は、TEXTDOMAINMAX (256) バイト以内に制限されています。

ロケール変更のために `setlocale(3C)` が呼び出されていない限り、マルチスレッドアプリケーションにおいて `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()`、`textdomain()`、および `bindtextdomain()` 関数を安全に使うことができます。

`gettext()`、`dgettext()`、`dcgettext()`、`textdomain()`、および `bindtextdomain()` 関数は Solaris メッセージカタログと GNU 互換メッセージカタログの両方を処理できます。`ngettext()`、`dngettext()`、`dcngettext()`、および `bind_textdomain_codeset()` 関数は GNU 互換メッセージカタログだけを処理できます。Solaris メッセージカタログと GNU 互換メッセージカタログの詳細については、`msgfmt(1)` のマニュアルページを参照してください。

ファイル

`/usr/lib/locale`

メッセージ・ドメイン・ファイルのデフォルトパス部分

`/usr/lib/locale/locale/LC_MESSAGES/domainname.mo`

言語 `locale` および `domainname` 用メッセージの入っているファイルのシステムデフォルト位置

`/usr/lib/locale/locale/LC_XXX/domainname.mo`

`LC_XXX` が `LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` である場合に、`dcgettext()` 呼び出し用の言語 `locale` および `domainname` メッセージの入っているファイルのシステムデフォルト位置

`dirname/locale/LC_MESSAGES/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の、ドメイン `domainname` およびパス部分 `dirname` 用メッセージが入っているファイルの位置

`dirname/locale/LC_XXX/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の `LC_XXX` が、`LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` のいずれかである場合 `dcgettext()` 呼び出し用のドメイン `domainname`、言語

## dgettext(3C)

*locale*、およびパス部分 *dirname* メッセージが入っているファイルの位置

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 `msgfmt(1)`, `xgettext(1)`, `iconv_open(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

## dngettext(3C)

名前	gettext, dgettext, dcgettext, ngettext, dngettext, dcngettext, textdomain, bindtextdomain, bind_textdomain_codeset – メッセージ処理関数
形式	
Solaris および GNU 互換	<pre>#include &lt;libintl.h&gt; char *gettext(const char *msgid); char *dgettext(const char *domainname, const char *msgid); char *textdomain(const char *domainname); char *bindtextdomain(const char *domainname, const char *dirname); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcgettext(const char *domainname, const char *msgid, int category);</pre>
GNU 互換	<pre>#include &lt;libintl.h&gt; char *ngettext(const char *msgid1, const char *msgid2, unsigned long int n); char *dngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n); char *bind_textdomain_codeset(const char *domainname, const char *codeset); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n, int category);</pre>
機能説明	<p>gettext(), dgettext(), dcgettext() 関数は、特定のドメイン内において、また、現在のロケールにおいて、指定の <i>msgid</i> 引数に基づいたターゲット文字列を取得します。gettext(), dgettext(), dcgettext() から返される文字列の長さは、これらの関数が呼び出されるまでは未定です。<i>msgid</i> 引数は、NULL で終わる文字列です。</p> <p>ngettext(), dngettext(), および dcngettext() 関数はそれぞれ、gettext(), dgettext(), および dcgettext() と同等ですが、GNU 互換メッセージカタログだけを処理することができ、複数形の処理方法が異なります。メッセージ文字列を検索するとき、ngettext(), dngettext(), および dcngettext() 関数は <i>msgid1</i> 引数を検索鍵として使用し、<i>n</i> 引数で複数形を決定します。メッセージカタログが見つからない場合、<i>n</i> == 1 であれば <i>msgid1</i> が返され、そうでなければ <i>msgid2</i> が返されます。</p> <p>環境変数 NLSPATH (environ(5) を参照) が、最初に LC_MESSAGES カタログの位置として検索されます。現在のロケールの LC_MESSAGES カテゴリの設定によって、gettext() および dgettext() が文字列取得に使用するロケールが決まります。</p>

*category* 引数は `dcgettext()` の使用するロケールを決めます。NLSPATH が定義されておらず、現在のロケールが C ロケールの場合、`gettext()`、`gettext()`、`dcgettext()` は、渡されたメッセージ文字列をそのまま返します。C 以外のロケールでは、NLSPATH が定義されていない場合、あるいは、NLSPATH で指定したコンポーネントでメッセージカタログが見つからない場合、メッセージカタログの検索には、次の段落で説明するスキームが使用されます。

LANGUAGE 環境変数は、使用する GNU 互換メッセージカタログを決定するために調べられます。LANGUAGE の値はコロン (:) で区切られたロケール名のリストです。LANGUAGE が定義されている場合、リストに指定されている順番で各ロケールが検索されます。そして、GNU 互換メッセージカタログが見つかった場合、そのメッセージが返されます。GNU 互換メッセージが見つかったが、対応する *msgid* が見つからない場合、*msgid* 文字列が返されます。LANGUAGE が定義されていない場合、あるいは、LANGUAGE の処理中に Solaris メッセージカタログが見つかったが GNU 互換メッセージカタログが見つからない場合、*dirname/locale/category/domainname.mo* というパス名でメッセージカタログが検索されます。ここで、*dirname* は `bindtextdomain()` で指定されたディレクトリ、*locale* はロケール名、*category* は LC\_MESSAGES または LC\_XXX のどちらかです。LC\_MESSAGES は `gettext()`、`dgettext()`、`ngettext()`、または `dngettext()` を呼び出した場合に使用されます。LC\_XXX は `dcgettext()` または `dcngettext()` を呼び出した場合に使用され、XXX 部分は *category* 引数で指定したロケールカテゴリ名と同じになります。

`gettext()` と `ngettext()` の場合、使用されるドメインは、最後に行なった `textdomain()` の有効な呼び出しによって設定されます。`textdomain()` の有効な呼び出しが行われていない場合は、デフォルト時のドメイン (`messages`) が使用されます。

`dgettext()`、`dngettext()`、`dcngettext()`、および `dcgettext()` の場合、使用されるドメインは *domainname* 引数で指定されます。この *domainname* 引数は、構文と意味において、`textdomain()` の *domainname* 引数と同等ですが、このドメインの選択が `dgettext()`、`dcgettext()`、`dngettext()` および `dcngettext()` 関数の呼び出し中にのみ有効であるという点で異なります。

`textdomain()` 関数は、アクティブな LC\_MESSAGES ロケールカテゴリの現在のドメイン名の設定 または照会を行います。*domainname* 引数は、正当なファイル名に使用できる文字のみからなる、NULL で終わる文字列です。

*domainname* 引数は、システム内の、一意のドメイン名です。1つのシステム内に、同一ドメインのバージョンが複数存在する場合は、`bindtextdomain()` を使用して名前空間の衝突を回避することができます。`textdomain()` を呼び出さない場合には、デフォルト時のドメインが選択されます。最後に行なった `textdomain()` の有効な呼び出しによるドメインの設定は、このあとに行う `setlocale(3C)` および `gettext()` の呼び出しにおいても有効です。

*domainname* 引数は、現在アクティブな、LC\_MESSAGES ロケールに適用されます。

## dngettext(3C)

*domainname* に NULL ポインタを設定して `textdomain()` を呼び出すことによって、ドメインの現在の状態に影響することなく、ドメインの現在の設定を照会することができます。*domainname* 引数に NULL 文字列を設定して `textdomain()` を呼び出すと、ドメインの設定は、デフォルト時のドメイン (`messages`) になります。

`bindtextdomain()` 関数は、メッセージドメインのパス部分 *domainname* と、*dirname* に入っている値を結合します。*domainname* が空文字列ではなく、まだ結合されていない場合には、`bindtextdomain()` は、*domainname* と *dirname* を結合します。

*domainname* が空文字列ではなく、すでに結合されている場合には、`bindtextdomain()` は、事前に行われていた結合部分を *dirname* に置き換えます。*dirname* 引数に使用できるのは、`gettext()`、`dgettext()`、または `dcgettext()` が呼び出されたときに解決される、絶対パス名または相対パス名です。*domainname* が NULL ポインタまたは空文字列の場合は、`bindtextdomain()` は NULL を返します。ユーザー定義のドメイン名は、`sys_` で始めることはできません。この文字列で始まるドメイン名は、システムで使用するために予約されています。

`bind_textdomain_codeset()` 関数を使用すると、ドメイン *domainname* のメッセージカタログ用の出力コードセットを指定できます。*codeset* 引数は、`iconv_open(3C)` 関数に対して使用可能な有効なコードセット名であるか、空ポインタである必要があります。*codeset* 引数が空ポインタの場合、`bind_textdomain_codeset()` はドメイン *domainname* 用に現在選択されているコードセットを返します。コードセットが選択されていない場合、`bind_textdomain_codeset()` は空ポインタを返します。`bind_textdomain_codeset()` 関数は複数回使用できます。同じ *domainname* 引数を指定して `bind_textdomain_codeset()` 関数を複数回使用した場合、先行する呼び出しで行われた設定は後続の呼び出しによって無効にされます。`bind_textdomain_codeset()` 関数は、選択されたコードセット名を含む文字列へのポインタを返します。この文字列は `bind_textdomain_codeset()` 関数に内部的に割り当てられているため、ユーザーが変更してはなりません。

戻り値

`gettext()`、`dgettext()`、および `dcgettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*msgid* 文字列を返します。

`ngettext()`、`dngettext()`、および `dcngettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*n* == 1 であれば、*msgid1* を返します。そうでなければ、*msgid2* を返します。

`gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` が返す文字列の個々のバイトには、空文字以外のあらゆる値が入ります。*msgid* が NULL ポインタである場合の戻り値は未定義です。返される文字列は、プログラムで修正してはならず、この後で `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` または `setlocale(3C)` を呼び出すことにより、無効になることがあります。`dgettext()`、`dcgettext()`、`dngettext()`、または `dcngettext()` に対する *domainname* 引数が NULL ポインタである場合、現在 `textdomain()` によってバインドされているドメインが使用されます。



## dngettext(3C)

`textdomain()` の正常な戻り値は、ドメインの現在の設定が入っている文字列へのポインタです。`domainname` が NULL ポインタの場合は、`textdomain()` は現在のドメインが入っている文字列へのポインタを返します。`textdomain()` を先に呼び出していない場合に `domainname` を空文字列にすると、デフォルト時のドメイン名が返されます。デフォルト時のドメイン名は、`messages` です。`textdomain()` が失敗した場合、NULL ポインタが返されます。

`bindtextdomain()` からの戻り値は、`dirname` が入った NULL で終わる文字列です。`dirname` が NULL の場合は、`domainname` と結び付けられたディレクトリバインディングです。何も結合されていない場合には、デフォルトの戻り値は `/usr/lib/locale` になります。`domainname` が NULL ポインタまたは空文字列の場合、`bindtextdomain()` は何も行わず、NULL ポインタを返します。返された文字列を呼び出し側で変更してはなりません。`bindtextdomain()` が失敗した場合、NULL ポインタが返されます。

使用法 これらのルーチンでは、メッセージの長さに制限はありません。ただし、テキスト `domainname` は、TEXTDOMAINMAX (256) バイト以内に制限されています。

ロケール変更のために `setlocale(3C)` が呼び出されていない限り、マルチスレッドアプリケーションにおいて `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()`、`textdomain()`、および `bindtextdomain()` 関数を安全に使うことができます。

`gettext()`、`dgettext()`、`dcgettext()`、`textdomain()`、および `bindtextdomain()` 関数は Solaris メッセージカタログと GNU 互換メッセージカタログの両方を処理できます。`ngettext()`、`dngettext()`、`dcngettext()`、および `bind_textdomain_codeset()` 関数は GNU 互換メッセージカタログだけを処理できます。Solaris メッセージカタログと GNU 互換メッセージカタログの詳細については、`msgfmt(1)` のマニュアルページを参照してください。

ファイル

`/usr/lib/locale`

メッセージ・ドメイン・ファイルのデフォルトパス部分

`/usr/lib/locale/locale/LC_MESSAGES/domainname.mo`

言語 `locale` および `domainname` 用メッセージの入っているファイルのシステムデフォルト位置

`/usr/lib/locale/locale/LC_XXX/domainname.mo`

`LC_XXX` が `LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` である場合に、`dcgettext()` 呼び出し用の言語 `locale` および `domainname` メッセージの入っているファイルのシステムデフォルト位置

`dirname/locale/LC_MESSAGES/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の、ドメイン `domainname` およびパス部分 `dirname` 用メッセージが入っているファイルの位置

`dirname/locale/LC_XXX/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の `LC_XXX` が、`LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` のいずれかである場合 `dcgettext()` 呼び出し用のドメイン `domainname`、言語

## dngettext(3C)

属性 *locale*、およびパス部分 *dirname* メッセージが入っているファイルの位置  
次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 `msgfmt(1)`, `xgettext(1)`, `iconv_open(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

名前 euclen, euccol, eucscol – EUC 文字のバイト長および表示幅の取得

形式 #include <euc.h>

```
int euclen(const unsigned char *s);
int euccol(const unsigned char *s);
int eucscol(const unsigned char *str);
```

機能説明 euclen() 関数は、*s* が指す拡張 UNIX コード (EUC) 文字 (単一シフト文字がある場合はこれも含む) の長さをバイト単位で返します。

euccol() 関数は、*s* が指す EUC 文字の画面カラム幅を返します。

eucscol() 関数は、*str* が指す EUC 文字列の画面カラム幅を返します。

euclen() と euccol() の各関数では、*s* は文字の最初のバイトを指します。このバイトを調べてコードセットを判別します。現在のロケール用の文字型テーブルを用いて、コードセットのバイト長と表示幅に関する情報を取得します。

使用法 これらの関数は EUC でのみ動作します。

これらの関数は `setlocale(3C)` がロケール変更のために呼び出されていない限り、マルチスレッドアプリケーション上で安全に使用できます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 `getwidth(3C)`, `setlocale(3C)`, `attributes(5)`

## euclen(3C)

名前 euclen, euccol, eucscol – EUC 文字のバイト長および表示幅の取得

形式 #include <euc.h>

```
int euclen(const unsigned char *s);  
int euccol(const unsigned char *s);  
int eucscol(const unsigned char *str);
```

機能説明 euclen() 関数は、*s* が指す拡張 UNIX コード (EUC) 文字 (単一シフト文字がある場合はこれも含む) の長さをバイト単位で返します。

euccol() 関数は、*s* が指す EUC 文字の画面カラム幅を返します。

eucscol() 関数は、*str* が指す EUC 文字列の画面カラム幅を返します。

euclen() と euccol() の各関数では、*s* は文字の最初のバイトを指します。このバイトを調べてコードセットを判別します。現在のロケール用の文字型テーブルを用いて、コードセットのバイト長と表示幅に関する情報を取得します。

使用法 これらの関数は EUC でのみ動作します。

これらの関数は setlocale(3C) がロケール変更のために呼び出されていない限り、マルチスレッドアプリケーション上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 getwidth(3C), setlocale(3C), attributes(5)

名前 euclen, euccol, eucscol – EUC 文字のバイト長および表示幅の取得

形式 #include <euc.h>

```
int euclen(const unsigned char *s);
int euccol(const unsigned char *s);
int eucscol(const unsigned char *str);
```

機能説明 euclen() 関数は、*s* が指す拡張 UNIX コード (EUC) 文字 (単一シフト文字がある場合はこれも含む) の長さをバイト単位で返します。

euccol() 関数は、*s* が指す EUC 文字の画面カラム幅を返します。

eucscol() 関数は、*str* が指す EUC 文字列の画面カラム幅を返します。

euclen() と euccol() の各関数では、*s* は文字の最初のバイトを指します。このバイトを調べてコードセットを判別します。現在のロケール用の文字型テーブルを用いて、コードセットのバイト長と表示幅に関する情報を取得します。

使用法 これらの関数は EUC でのみ動作します。

これらの関数は `setlocale(3C)` がロケール変更のために呼び出されていない限り、マルチスレッドアプリケーション上で安全に使用できます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 `getwidth(3C)`, `setlocale(3C)`, `attributes(5)`

## fgetwc(3C)

名前	fgetwc – ストリームからワイド文字コードを読み込む
形式	<pre>#include &lt;stdio.h&gt; #include &lt;wchar.h&gt;  wint_t fgetwc(FILE*stream);</pre>
機能説明	<p>fgetwc() 関数は、<i>stream</i> が指す入力ストリームから次の文字 (存在する場合) を読み込み、対応するワイド文字コードに変換し、ストリーム用の該当するファイルポジション表示 (定義されている場合) を進めます。</p> <p>エラーが発生した場合、ストリーム用のファイルポジション表示の値がどうなるかは予測できません。</p> <p>fgetwc() 関数は、<i>stream</i> に対応するファイルの <code>st_atime</code> フィールドを、更新用にマークします。fgetwc(), fgetc(3C), fgets(3C), fgetws(3C), fread(3C), fscanf(3C),getc(3C),getchar(3C),gets(3C)、および scanf(3C) のいずれかの関数のうち、以前の ungetc(3C) または ungetwc(3C) 呼び出しで返されなかったデータを返す <i>stream</i> を使ったものが最初に正常終了したとき、<code>st_atime</code> フィールドは更新用にマークされます。</p>
戻り値	<p>正常終了時、fgetwc() 関数は <i>stream</i> が指す入力ストリームから得た文字のワイド文字コードを <code>wint_t</code> 型に変換して返します。</p> <p>ストリームがファイルの終わり (EOF 状態) にあるとき、ストリーム用の EOF 表示が設定されて fgetwc() は WEOF を返します。</p> <p>読み込みエラーが発生すると、ストリーム用のエラー表示が設定され、fgetwc() は WEOF を返して <code>errno</code> をエラーを示す値に設定します。</p>
エラー	<p>データを読み込む必要があるとき、以下の条件で fgetwc() は異常終了します。</p> <p>EAGAIN            <i>stream</i> に対応するファイル記述子に <code>O_NONBLOCK</code> フラグが設定されていて、プロセスの fgetwc() 動作が待たされる可能性がある</p> <p>EBADF            <i>stream</i> に対応するファイル記述子が、読み込み用にオープンされた正しいものでない</p> <p>EINTR            シグナルを受け取り、読み込みが終了した。データは転送されていない</p> <p>EIO              物理的な入出力エラーが発生した。または、プロセスがバックグラウンドプロセスのグループでそのグループが制御端末からの読み込みをしようとして、さらにプロセスが SIGTTIN シグナルを無視またはブロックしているか、あるいはプロセスグループの親がなくなっている</p> <p>EOVERFLOW       ファイルは通常ファイルで、対応する <i>stream</i> に関連するオフセット最大値、またはそれ以上で作成しようとした</p> <p>fgetwc() 関数は以下の条件で異常終了します。</p> <p>ENOMEM           使用可能な記憶領域が十分でない</p>

- ENXIO 存在しないデバイスに対して要求が出された、またはデバイスの能力を超えた要求が出された
- EILSEQ 入力ストリームから得たデータが、正しい文字を構成していない
- 使用法 エラー状態と EOF 状態とを区別するには、`ferror(3C)` と `feof(3C)` 関数を使用してください。
- 属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

- 関連項目 `setlocale(3C)`, `feof(3C)`, `ferror(3C)`, `fgetc(3C)`, `fgets(3C)`, `fgetws(3C)`, `fopen(3C)`, `fread(3C)`, `fscanf(3C)`, `getc(3C)`, `getchar(3C)`, `gets(3C)`, `scanf(3C)`, `ungetc(3C)`, `ungetwc(3C)`, `attributes(5)`

## fgetws(3C)

名前 getws, fgetws – ストリームからワイド文字列を取得する

形式

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

wchar_t *getws(wchar_t *ws);

#include <stdio.h>
#include <wchar.h>

wchar_t *fgetws(wchar_t *ws, int n, FILE *stream);
```

機能説明 getws() 関数は、標準入力ストリーム stdin から文字列を読み取り、それをワイド文字に変換し、ws が指す配列に書き込みます。この動作は、復帰改行文字が読み取られて変換され、ws に送られるまで、またはファイルの終端状態になるまで続けられます。ワイド文字列 ws は、ワイド文字コードの NULL 文字で終了します。

fgetws() 関数は、stream から文字を読み取り、それをワイド文字に変換し、ws が指す wchar\_t 配列に書き込みます。この動作は、n-1 文字まで読み取られるか、復帰改行文字が読み取られて変換され、ws に送られるまで、またはファイルの終端状態になるまで続けられます。ワイド文字列 ws は、ワイド文字コードの NULL 文字で終了します。

エラーが発生すると、ストリームに対するファイル指示子の戻り値が仲介になります。

fgetws() 関数は、stream に関連づけられているファイルの st\_atime フィールドに、更新のマークを付けます。fgetc(3C)、fgets(3C)、fgetwc(3C)、fgetws()、fread(3C)、fscanf(3C)、getc(3C)、getchar(3C)、gets(3C)、scanf(3C) の実行が最初に成功したときに、直前の scanf(3C) への呼び出しまたは scanf(3C) によって与えられないデータを返す stream を使用して、st\_atime フィールドに更新のマークが付けられます。

戻り値 getws() および fgetws() は、実行が成功すると ws を返します。ストリームがファイルの終端である場合、そのストリームについてファイル終端を示す指示子が設定され、fgetws() 関数が NULL ポインタを返します。読み取りエラーが発生した場合、ストリームに対してエラー指示子が設定され、fgetws() が NULL ポインタを返し、エラーを示す errno を設定します。

エラー fgetws() が失敗する状態については、fgetwc(3C) を参照してください。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 ferror(3C)、fgetwc(3C)、fread(3C)、getwc(3C)、putws(3C)、scanf(3C)、attributes(5)



名前 printf, fprintf, sprintf, snprintf – 書式付き出力

形式 #include <stdio.h>

```
int printf(const char *format, /* args*/ ...);
int fprintf(FILE *stream, const char *format, /* args*/ ...);
int sprintf(char *s, const char *format, /* args*/ ...);
int snprintf(char *s, size_t n, const char *format, /* args*/ ...);
```

機能説明 printf() 関数は、標準出力ストリーム stdout 上に出力します。

fprintf() 関数は、出力ストリーム *stream* 上に出力します。

sprintf() 関数は、*s* から始まる連続したバイトを出力し、最後に NULL バイト (\0) をつけます。ユーザーは、十分な記憶領域が利用できることを確認する必要があります。

snprintf() 関数は、sprintf() に引数 *n* を追加したのと同様です。*n* には *s* によって参照されるバッファサイズが指定されます。バッファは、常に NULL バイトで終了します。

これらの各関数は、*format* の制御下で各引数の変換、書式化および出力を行います。*format* とは、初期シフトの状態 (もしあれば) で始まるか、または終わる文字列のことです。*format* は、以下に示す 0 個以上の指示語で構成されます。

- 通常文字。出力ストリームに単純にコピーされる
- 変換指定。各指定の結果として 0 個以上の引数を取り出す

*format* 引数が不十分である場合、結果は未定義です。引数が残っていて、*format* が使い果たされた場合、余分な引数は評価されますが、評価されたとしても無効になります。

変換は、引数リストの *format* のあとに続く、次の未使用の引数にではなく、*n* 番目の引数に適用されます。この場合、変換文字 % (後述を参照) は %*n*\$ シーケンスに置換されます。ここで *n* は [1, NL\_ARGMAX] の範囲の 10 進数の整数で、引数リストにある引数の位置を示します。この機能によって、特定の言語に適切な順序で引数を選択する、書式文字列を定義できます (「使用例」参照)。

書式文字列に、変換指定の書式 %*n*\$ が含まれる場合、引数リスト中の番号付けされた引数は、要求された回数だけ書式文字列から参照されます。

書式文字列に、変換指定の書式 % が含まれる場合、引数リスト中の各引数は 1 度だけ使用されます。

printf() 関数のすべての形式において、言語依存の小数点文字を出力文字列に挿入できます。小数点文字は、プログラムのロケール (LC\_NUMERIC カテゴリ) によって定義されます。POSIX ロケールおよび小数点文字が定義されていないロケールでは、小数点文字のデフォルトはピリオド (.) です。

## fprintf(3C)

- 変換指定 各変換指定は % 文字、または %n\$ 文字シーケンスによって導入されます。これらの文字の後には、以下に示す (順番で) 文字列が続きます。
- 変換すべき次の引数を指定する、最後に \$ が付いた 10 進数の文字列のフィールド (任意)。このフィールドがない場合は、最後に変換された引数の次の *args* が変換されます。
  - 変換指定の意味を変更する 0 個以上の *flags* (順不同)。
  - *field width* の最小値 (任意)。変換された値がフィールド幅より少ないバイトである場合は、デフォルトとしてフィールド幅の左側に (以下で説明する左位置合わせフラグ (-) が指定されている場合は右側に)、空白をパディング (文字詰め) します。フィールド幅は、アスタリスク (\*) (以下に説明) または 10 進数の整数の形式を取ります。

変換文字が *s* の場合、標準準拠のアプリケーション (standards(5) を参照) は、フィールド幅を、出力される場合の最小バイト数として解釈します。また、標準準拠ではないアプリケーションは、フィールド幅を、画面表示の最小カラム数として解釈します。標準準拠ではないアプリケーションに対しては、たとえば %10s とは、変換された値が 7 カラムの画面幅の場合は 右側に 3 個分の空白がパディングされることを意味します。

書式が %ws ならば、フィールド幅を画面表示の最小カラム数として解釈します。
  - 精度を指定する *precision* (任意)。d、i、o、u、x、および X 変換の場合、数値の最小桁数を表します。この場合、このフィールドの先頭は 0 でパディングされます。e、E、および f 変換の場合、小数点文字以下の数字の桁数を表します。g および G 変換の場合、最大有効桁数を表します。また、s および S 変換の文字列からは、最大バイト数が出力されます。精度は、最初にピリオド (.) が来て、そのあとにアスタリスク (\*) (以下に説明) または 10 進数の文字列 (任意) が続く形式になります。NULL の 10 進数の文字列は 0 として扱われます。精度がその他の変換文字で表される場合、動作は未定義です。

変換文字が *s* または *S* の場合、標準準拠のアプリケーション (standards(5) を参照) は、精度を、出力される場合の最大バイト数として解釈します。また、標準準拠ではないアプリケーションは、精度を、画面表示の最大カラム数として解釈します。標準準拠ではないアプリケーションに対しては、たとえば %.5s は、画面の 5 カラムに表示する文字列部分だけを出力します。完全な文字だけが書き込まれません。

%ws の場合、精度を画面表示の最大カラム数として解釈します。精度は、最初にピリオドが来て、その後 10 進数の文字列が続く形式になります。NULL の 10 進数の文字列は 0 として扱われます。精度で指定したパディングは、フィールド幅で指定したパディングより優先されます。
  - h (任意)。変換文字 d、i、o、u、x、および X が short int 型または unsigned short int 型の引数に適用されることを指定します (引数は、整数変換 (インテグラルプロモーション) の規則に従って変換されます。値は short int または unsigned short int の形式に変換されてから表示されます)。また h は、変換文字 n が short int 型の引数へのポインタに適用されることを指定します。l (小文字のエル) (任意指定) は、変換文字 d、i、o、u、x、および X が long int 型または unsigned long int 型の引数に適用されることを指定します。また l (小文字のエル) は、変換文字 n が long int 型の引数へのポインタに適用されることを指定します。ll (小文字のエルが 2 つ) (任意指定) は、変換文字

d、i、o、u、x、および X が long long 型または unsigned long long 型の引数に適用されることを指定します。また ll (小文字のエルが 2 つ) は、変換文字 n が long long 型の引数へのポインタに適用されることを指定します。L (任意) は、変換文字 e、E、f、g、および G が long double 型の引数に適用されることを指定します。h、l、および L が上記以外の変換文字に現われたときの変換結果は、未定義です。

- l (小文字のエル) (任意)。変換文字 c が wint\_t 引数に適用されることを指定します。また l (小文字のエル) は、変換文字 s が wint\_t 引数へのポインタに適用されることを指定します。
- 適用される変換の型を意味する *conversion character* (下記参照)。

フィールド幅および精度の両方またはどちらか一方は、アスタリスク (\*) で示すこともできます。この場合、int 型の引数によってフィールド幅または精度を指定します。フィールド幅および精度の両方またはどちらか一方を指定する引数 (もしあれば) は、変換される順番で引数の前に現れなければなりません。フィールド幅が負の値の場合、- フラグの後に正のフィールド幅を指定したように扱われます。精度が負の値の場合は、精度を省略したように扱われます。%n\$ という形式で変換を指定している書式文字列では、フィールド幅または精度は \*m\$ シーケンスで示すことができます。この場合 m は、[1, NL\_ARGMAX] の範囲の 10 進数の整数で、整数引数の引数リスト内 (書式引数に続く) のフィールド幅または精度の位置を示します。以下に例を示します。

```
printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

*format* には、番号付けされた引数指定 (つまり %n\$ と \*m\$)、または番号付けされていない引数指定 (つまり % と \*) のいずれかを含むことができます。通常、両方を指定することはできません。唯一の例外として、%% と %n\$ の形式の両方を含むことがあります。*format* 文字列に、番号付けされた引数と番号付けされていない引数とを、同時に指定した場合の結果は未定義です。番号付けされた引数を指定するとき、N 番目の引数を指定する場合は、最初から (N-1) 番目までの先行引数を書式文字列内で指定する必要があります。

フラグ文字 以下に、フラグ文字とその意味を説明します。

- ' 10 進数変換 (%i、%d、%u、%f、%g、または %G) の結果の整数部分が、千単位のグループ化文字でフォーマットされます。その他の変換についての動作結果は未定義です。貨幣以外のグループ化文字が使用されます。
- 変換の結果は、フィールド内で左詰めされます。このフラグを指定しない場合、変換の結果は右詰めされます。
- + 符号つき変換の結果は、常に符号 (+ または -) で始まります。このフラグを指定しない場合、負の値を変換するときだけ符号で始まります。
- 空白 (スペース) 符号つき変換の最初の文字が符号でない場合または変換の結果に文字がない場合は、結果の前に空白がおかれます。つまり、空白のフラグと + フラグを共に指定した場合、空白のフラグは無視されます。
- # 値は代替形式に変換されます。c、d、i、s、および u 変換の場合、このフラグは影響しません。o 変換の場合、このフラグは (必要な場合には) 精度をあげて、結果の最初の数字をゼロにします。x (または X) 変換では、0

## fprintf(3C)

	<p>以外の結果の前に 0x (または 0X) が追加されます。e、E、f、g、および G 変換の場合、必ず、結果は小数点付き数字になります。小数点の後に数字が続かない場合でも小数点が付きます (このフラグが指定されていない場合、これらの変換の結果では、小数点以降の数字がある場合のみ小数点がつけられます)。g および G 変換の場合、結果から後続 0 が削除されません (通常は削除されます)。</p>
0	<p>d, i, o, u, x, X, e, E, f, g, および G 変換の場合、先行 0 を (符号または基数の後ろに) 用いて、フィールド幅をパディングします。空白がパディングされることはありません。0 フラグおよび - フラグをともに指定すると、0 フラグが無視されます。d、i、o、u、x、および X の各変換では、精度を指定すると、0 フラグは無視されます。0 フラグおよび ' フラグをともに指定すると、0 パディングの前に文字が挿入されます。他の変換では、このフラグの動作は未定義です。</p>
変換文字	<p>各変換文字を指定すると、0 個以上の引数が取り出されます。書式に関する引数の指定が不十分な場合の変換結果は、未定義です。書式が終了しても引数が残っている場合は、残りの引数は無視されます。</p> <p>変換文字とその意味を以下で説明します。</p>
d,i	<p>int 引数は、[-]dddd の形式で符号付き 10 進数に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。</p>
o	<p>unsigned int 引数は、dddd の形式で符号なし 8 進数の書式に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。</p>
u	<p>unsigned int 引数は、dddd の形式で符号なし 10 進数の書式に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。</p>
x	<p>unsigned int 引数は、dddd の形式で符号なし 16 進数の書式に変換されます。abcdef 文字が使用されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表せる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。</p>
X	<p>ABCDEF 文字が abcdef 文字の代わりに使用される点を除いて、x 変換文字と同じ動作です。</p>
f	<p>double の引数は、[-]ddd.ddd の形式で 10 進数に変換されます。小数点付き数字 setlocale(3C 参照) の後の桁数は、精度で指定した数です。精度が指定されていない場合は、6 とみなされます。精度を明示的に 0 に指定</p>

	し、# フラグを指定しない場合、小数点付き数字は出力されません。小数点付き数字が出力されると、1 個以上の数字がその前に付きます。値は適切な桁数に丸められます。
e,E	double の引数は、[-]d.ddde±dd の形式に変換されます。小数点付き数字の前には 1 個の数字が付きます (引数が 0 でない場合はこの数字も 0 ではありません)。小数点付き数字の後には、精度で指定した数の数字が続きます。精度を指定しない場合は、6 とみなされます。精度が 0 で、# フラグを指定しない場合、小数点付き数字は出力されません。E 変換文字を使用すると、e ではなく E を数字につけて指数を示します。指数は、必ず 2 桁以上の数字です。値は適切な桁数に丸められます。
g,G	double の引数は、f または e の形式 (G 変換文字の場合は E 形式) で出力されます。精度は有効桁数を示します。明示的な精度が 0 である場合は、有効桁数が 1 桁になります。出力形式は変換される値によって決まります。変換の結果出力される指数が -4 より小さい場合、または精度以上の場合だけ、e (または E) 形式で出力されます。結果の小数部分からは後続 0 が取り除かれます。小数点付き数字は、その後に数字が続く場合だけ出力されます。
c	int の引数は、unsigned char 符号なし文字に変換され、結果のバイトが出力されます。  l (小文字のエル) を修飾子として指定した場合、wint_t 引数は、精度は付けなくて、wchar_t 型の 2 つの要素の配列をポイントする引数を付けた ls 変換指定を行なった場合と同じように変換されます。最初の要素には ls 変換指定への wint_t 引数を含み、2 番目の要素には NULL ワイド文字を含みます。
C	lc と同じです。
wc	int の引数はワイド文字 (wchar_t) に変換されて出力されます。
s	引数は char 配列へのポインタにする必要があります。配列の中身が最後の NULL バイトまで書き込まれます (NULL バイトは含まない)。精度を指定すると、標準準拠のアプリケーション (standards(5) を参照) は、精度によって指定されるバイト数だけを書き込みます。また、標準準拠ではないアプリケーションは、精度によって指定された表示画面の カラム数に表示される文字列部分だけを出力します。精度を指定しないと場合は、無限とみなされ、最初の NULL バイトまでのすべてのバイトが出力されます。値が NULL の引数の変換は、未定義の結果になります。  修飾子として l (小文字のエル) を指定した場合、引数は wchar_t 型の並びへのポインタである必要があります。その並びからのワイド文字コードは、NULL ワイド文字コードに至るまでで、末尾に NULL ワイド文字コードを含まない文字に変換されます (最初のワイド文字が変換される前に、mbstate_t オブジェクトを 0 に初期化して記述された変換状態の、wrtomb(3C) 関数への呼び出しによる結果と同じように変換されます)。結果として生じる文字は、末尾の NULL 文字 (バイト) に至るまで (ただし、末尾には含まない) で書き込まれます。精度を指定しない場合、

## fprintf(3C)

	配列には、NULL ワイド文字を含む必要があります。精度を指定した場合、指定した文字 (バイト) 以上は書き込まれません (もしあれば、シフトシーケンスを含む)。配列が NULL ワイド文字を含む必要がある場合は、精度によって指定された文字シーケンスの長さを書き込み、関数は、一度配列の最後を過ぎたワイド文字へはアクセスを必要としません。文字の一部を書き込むことはありません。
s	ls と同じです。
ws	引数は wchar_t 配列へのポインタの文字列にする必要があります。その配列の中身が最後の NULL 文字まで書き込まれます (NULL 文字は含まない)。精度を指定すると、精度によって指定された表示画面のカラム数に表示するワイド文字列部分だけを出力します。精度を指定しない場合は無限とみなされ、最初の NULL 文字までのすべてのワイド文字が出力されます。値が NULL の引数を指定した場合の変換は、未定義の結果になります。
p	引数は void へのポインタにする必要があります。ポインタの値は、出力可能なシーケンスに変換されます。これらの文字は、scanf(3C) 関数の %p 変換で一致した文字と同じ文字である必要があります。
n	引数は、printf() 関数のうちの 1 つの呼び出しにおいて、この変換文字が指定されるまでに、出力先に書き込まれたバイト数が格納される整数へのポインタです。引数は変換されません。
%	% を出力します。引数は変換されません。全体の変換を指定するには %% にしてください。
	変換指定が上記の形式のいずれにも当てはまらない場合、変換の結果は未定義となります。
	浮動小数点値によって「無限」を内部的に表現している場合は、[±]Infinity が出力されます。希望した出力文字列の長さに応じて、Infinity は Infinity または Inf のどちらかになります。符号は、前述の規則に従って出力されます。
	浮動小数点値によって「数でない値」を内部的に表現している場合は、[±]NaN が出力されます。符号は、前述の規則に従って出力されます。
	フィールド幅が存在していなかったり、あるいはその値が小さい場合でも、フィールドが切り捨てられることはありません。変換の結果がフィールド幅よりも広い場合は、その結果を収容できるようにフィールドが拡張されます。printf() および fprintf() によって生成される文字は、putc(3C) 関数を呼び出した場合と同じように出力されます。
	ファイルの st_ctime と st_mtime フィールドは、printf() または fprintf() の正常実行への呼び出しと、同じストリーム上の fflush(3C) または fclose(3C) あるいは exit(3C) または abort(3C) への呼び出しへの次の正常終了の呼び出しとの間で、更新するためにマークされます。
戻り値	printf(), fprintf(), および sprintf() 関数は、転送されるバイト数 (sprintf() の場合は最後の NULL バイトを除く) を返します。

## fprintf(3C)

	<p>snprintf() 関数はフォーマットされた文字数、つまり (バッファに十分な大きさがある場合に) バッファに書き込まれた文字数を返します。snprintf() への呼び出し時に <i>n</i> の値が 0 である場合、1 未満の指定されていない値が返されます。</p> <p>出力エラーが検出された場合、各関数は負の値を返します。</p>
エラー	<p>printf() と fprintf() の両関数が異常終了する、または異常終了する可能性がある条件については、fputc(3C) または fputwc(3C) を参照してください。</p> <p>さらに printf() のすべての形式において、以下の条件で異常終了します。</p> <p>EILSEQ            有効な文字に対応しないワイド文字コードが検出された。</p> <p>EINVAL            引数が足りない</p> <p>さらに printf() と fprintf() は、以下の条件で異常終了します。</p> <p>ENOMEM            使用可能な記憶領域が不足している</p>
使用法	<p>printf() 関数の呼び出しに wint_t または wchar_t 型のオブジェクトが存在する場合、これらのオブジェクトを定義するヘッダー &lt;wchar.h&gt; も含む必要があります。</p> <p>sprintf() と snprintf() の両関数はマルチスレッドアプリケーションでは MT-Safe です。ロケール変更のために setlocale(3C) が呼び出されていない限り、マルチスレッドアプリケーションにおいて printf() および fprintf() 関数を安全に使用できます。</p>
エスケープシーケンス	<p>通常 printf() 関数に対する書式化文字列を入力するときには、C 言語に組み込まれている以下のエスケープシーケンスを使用します。ただし、これらのエスケープシーケンスは、printf() 関数によってではなく C コンパイラによって処理されます。</p> <p>\a            ベルを鳴らして警報を出します。</p> <p>\b            バックスペース。現在位置が行頭でなければ、出力位置を現在位置の 1 文字前に移動させます。</p> <p>\f            フォームフィード。出力位置を次の論理ページの最初の出力位置に移動させます。</p> <p>\n            改行。出力位置を次の行の行頭に移動させます。</p> <p>\r            キャリッジリターン。出力位置を現在行の行頭に移動させます。</p> <p>\t            水平タブ。出力位置を、現在行上の次の水平ハードタブ位置に移動させます。</p> <p>\v            垂直タブ。出力位置を、垂直ハードタブ位置の始めに移動させます。</p> <p>また C 言語では、次の形式の文字シーケンスがサポートされています。</p> <p>\octal-number および</p>

## fprintf(3C)

\hex-number8 進数 octal-number または 16 進数 hex-number で表す文字に変換します。たとえば、ASCII 表現の場合、文字 a は \141 として書き込まれ、文字 Z は \132 として書き込まれます。この文法は、NULL 文字 \0 を表す場合に最もよく使用されます。これは定数ゼロ (0) とまったく同等です。通常の 8 進数と同様に、8 進数に接頭辞ゼロが含まれていないことに注意してください。16 進数を指定するには、接頭辞の 0 を取り除いて接頭辞が x (小文字) になるようにします (大文字の X は不可です)。16 進シーケンスのサポートは、ANSI の拡張機能です。standards(5) を参照してください。

**使用例** 例 1 言語に依存する日付と時刻の形式を出力するには、以下のステートメントを使用します。

```
printf (format, weekday, month, day, hour, min);
```

米国の使用法では、*format* は次の文字列へのポインタになります。

```
"%s, %s %d, %d:%.2d\n"
```

生成されるメッセージは次のとおりです。

```
Sunday, July 3, 10:02
```

ドイツの使用法では、*format* は次の文字列へのポインタになります。

```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

生成されるメッセージは次のとおりです。

```
Sonntag, 3. Juli, 10:02
```

例 2 `Sunday, July 3, 10:02` の形式で日付と時刻を出力し、`weekday` と `month` が NULL で終わる文字列へのポインタにする場合は以下のようにになります。

```
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);
```

例 3 小数点以下 5 桁まで出力する場合は以下のようにになります。

```
printf("pi = %.5f", 4 * atan(1.0));
```

**デフォルト** 例 4 標準準拠ではないアプリケーション (standards(5) を参照) での `printf()` の動作だけに適用します。20 文字幅で名前のリストを表示する場合は以下のようにになります。

```
printf("%20s%20s%20s", lastname, firstname, middlename);
```

**属性** 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み



fprintf(3C)

関連項目 | exit(2), lseek(2), write(2), abort(3C), ecvt(3C), exit(3C), fclose(3C),  
fflush(3C), fputwc(3C), putc(3C), scanf(3C), setlocale(3C), stdio(3C),  
wcstombs(3C), wctomb(3C), attributes(5), environ(5), standards(5)

## fputc(3C)

名前	fputc, putwc, putwchar – ワイド文字コードをストリームへ書き出す						
形式	<pre>#include &lt;stdio.h&gt; #include &lt;wchar.h&gt;  wint_t fputc(wchar_t wc, FILE*stream); wint_t putwc(wchar_t wc, FILE*stream);  #include &lt;wchar.h&gt; wint_t putwchar(wchar_t wc);</pre>						
機能説明	<p><b>fputc()</b> fputc() 関数は、ワイド文字コード <i>wc</i> に対応する文字を、<i>stream</i> が指す出力ストリーム中に書き出します。書き出す位置は、出力ストリーム用のファイルポジション表示 (定義されていれば) が指す位置です。出力後、ファイルポジション表示の値を進めます。ファイルが位置指定要求をサポートしていない場合、またはストリームが追加モードでオープンされていた場合には、文字は出力ストリームに追加されます。書き込み中にエラーが発生すると、出力ファイルのシフト状態は未定義のままになります。</p> <p>ファイル中の <i>st_ctime</i> と <i>st_mtime</i> の両フィールドは、fputc() が正常終了してから、同じストリームに対する次の fflush(3C) または fclose(3C) の呼び出しが正常終了するまで、もしくは exit(2) または abort(3C) が呼び出されるまでの間に更新されるようマークされます。</p> <p><b>putwc()</b> putwc() 関数は、マクロとして実装される点を除いては、fputc() 関数と同じです。</p> <p><b>putwchar()</b> putwchar(<i>wc</i>) の呼び出しは、putwc(<i>wc</i>, stdout) と同じです。putwchar() ルーチンはマクロとして実装されます。</p>						
戻り値	正常終了時、fputc()、putwc()、および putwchar() は <i>wc</i> を返します。正常終了しない場合は WEOF を返します。このとき、ストリームのエラー表示がセットされ、errno はエラーを示す値に設定されます。						
エラー	ストリームが蓄積されていないか、あるいはストリームのバッファ中のデータを書き出す必要があるとき、fputc()、putwc()、および putwchar() 関数は以下の条件で異常終了します。						
	<table border="0"> <tr> <td style="vertical-align: top;">EAGAIN</td> <td><i>stream</i> に対応するファイル記述子に O_NONBLOCK フラグが設定されていて、プロセスの動作が書き込みで待たされる可能性がある</td> </tr> <tr> <td style="vertical-align: top;">EBADF</td> <td><i>stream</i> に対応するファイル記述子が、書き込み用にオープンされた正しいものでない。または、ファイルは通常ファイルで、対応する <i>stream</i> に関連するオフセット最大値、またはそれ以上で作成しようとした</td> </tr> <tr> <td style="vertical-align: top;">EINTR</td> <td>シグナルを受け取り、書き込みが終了した。データは転送されていない</td> </tr> </table>	EAGAIN	<i>stream</i> に対応するファイル記述子に O_NONBLOCK フラグが設定されていて、プロセスの動作が書き込みで待たされる可能性がある	EBADF	<i>stream</i> に対応するファイル記述子が、書き込み用にオープンされた正しいものでない。または、ファイルは通常ファイルで、対応する <i>stream</i> に関連するオフセット最大値、またはそれ以上で作成しようとした	EINTR	シグナルを受け取り、書き込みが終了した。データは転送されていない
EAGAIN	<i>stream</i> に対応するファイル記述子に O_NONBLOCK フラグが設定されていて、プロセスの動作が書き込みで待たされる可能性がある						
EBADF	<i>stream</i> に対応するファイル記述子が、書き込み用にオープンされた正しいものでない。または、ファイルは通常ファイルで、対応する <i>stream</i> に関連するオフセット最大値、またはそれ以上で作成しようとした						
EINTR	シグナルを受け取り、書き込みが終了した。データは転送されていない						

## fputc(3C)

- EIO                   物理 I/O エラーが発生した。または、プロセスがバックグラウンドプロセスのグループで、そのグループが制御端末に書き込もうとしていて、TOSTOP が設定されており、プロセスが SIGTTOU を無視もブロックもせず、さらにプロセスグループの親がなくなっている
- ENOSPC               ファイルが存在するデバイス上に、空き領域がない
- EPIPE                プロセスによる読み込み用にオープンされていないパイプまたは FIFO へ書き込もうとした。この場合、SIGPIPE シグナルもプロセスへ送られる

fputc(), putwc(), および putwchar() 関数は以下の条件で異常終了します。

- ENOMEM              使用可能な記憶領域が十分でない
- ENXIO                存在しないデバイスに対して要求が出された、またはデバイスの能力を超えた要求が出された
- EILSEQ               ワイド文字コード *wc* が、正当な文字に対応していない

使用法   putwc() および putwchar() のマクロには、関数が存在します。関数のフォームを得るには、マクロ名を未定義にする必要があります (たとえば #undef putc)。

マクロのフォームを使用した場合、putwc() は *stream* 引数を 2 回以上、評価します。特に putwc(*wc*, \*f++) は正しく動作しません。*stream* 引数に影響するような評価をする場合は、代わりに fputc() 関数を使用するようにしてください。

属性    次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目   exit(2), ulimit(2), abort(3C), fclose(3C), ferror(3C), fflush(3C), fopen(3C), setbuf(3C), attributes(5)

## fputws(3C)

名前 fputws – ワイド文字列をストリームへ書き出す

形式 

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

int fputws(const wchar_t *s, FILE *stream);
```

機能説明 fputws() 関数は、*ws* が指すワイド文字列 (NULL で終わるもの) に対応する文字列を、*stream* が指すストリームに書き出します。最後の NULL ワイド文字コードに対応する文字の出力や 改行文字も付加されません。

ファイル中の `st_ctime` と `st_mtime` の両フィールドは、`fputws()` が正常終了してから、同じストリームに対する次の `fflush(3C)` または `fclose(3C)` の呼び出しが正常終了するまで、もしくは `exit(2)` または `abort(3C)` が呼び出されるまでの間に更新されるようマークされます。

戻り値 正常終了時、`fputws()` は負でない値を返します。正常終了しない場合には、`-1` を返し、ストリームのエラー表示を設定します。このとき `errno` はエラーを示す値に設定されます。

エラー `fputwc(3C)` の説明を参照してください。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 `exit(2)`, `abort(3C)`, `fclose(3C)`, `fflush(3C)`, `fopen(3C)`, `fputwc(3C)`, `attributes(5)`

名前 scanf, fscanf, sscanf, vscanf, vfscanf, vsscanf – 書式付き入力の変換

形式

```
#include <stdio.h>

int scanf(const char *format, ...);

int fscanf(FILE*stream, const char *format, ...);

int sscanf(const char *s, const char *format, ...);

#include <stdarg.h>
#include <stdio.h>

int vscanf(const char *format, va_list arg);

int vfscanf(FILE *stream, const char *format, va_list arg);

int vsscanf(const char *s, const char *format, va_list arg);
```

機能説明

scanf() 関数は、標準入力ストリーム stdin を読み取ります。

fscanf() 関数は、入力ストリーム *stream* を読み取ります。

sscanf() 関数は、文字列 *s* を読み取ります。

vscanf()、vfscanf()、および vsscanf() 関数はそれぞれ、scanf()、fscanf()、および sscanf() と同等ですが、これらの関数を呼び出すときは、可変数個の引数ではなく、<stdarg.h> ヘッダーで定義されている引数リスト (stdarg(3HEAD) のマニュアルページを参照) を使用します。これらの関数は va\_end() マクロを呼び出しません。そのため、アプリケーションは、これらの関数を使用した後、va\_end(*ap*) を呼び出してクリーンアップ作業を行う必要があります。

これらの関数は、それぞれ、バイトを読み取り、書式にしたがってそれを解釈し、その結果を引数に格納します。これらの関数には、引数として、制御文字列 *format* (以下で説明)、および変換された入力の格納場所を示す *pointer* 群を指定します。書式に対して十分な引数が指定されていない場合、変換の結果は未定義となります。書式が終了しても引数が残っている場合、残りの引数は評価されますが、評価されたとしても無視されます。

変換は、次の未使用の引数にではなく、引数リストの *format* のあとに続く *n* 番目の引数に適用されます。この場合、変換文字 % (後述を参照) は %*n*\$ シーケンスに置換されます。ここで *n* は [1, NL\_ARGMAX] の範囲の 10 進数の整数です。この変換機能は、特定の言語に合わせた順序で引数を選択するように、書式文字列の定義を規定します。書式文字列に変換指定の書式 %*n*\$ が含まれる場合、引数リスト中の番号付けされた引数が、複数回、書式文字列から参照されるかどうかについては不定です。

*format* には、変換指定の形式、つまり % または %*n*\$ のいずれかを含むことができます。ただし、通常は 1 つの *format* 文字列に % と %*n*\$ の両方を指定することはできません。唯一の例外として、%% または %\* を %*n*\$ の形式に入れることができます。

## fscanf(3C)

`fscanf()` 関数のすべての形式において、入力文字列中の言語依存の小数点文字を検出できます。小数点文字は、プログラムのロケール (LC\_NUMERIC カテゴリ) によって定義されます。POSIX ロケール および小数点文字が定義されていないロケールでは、小数点文字のデフォルトはピリオド (.) です。

*format* とは、初期シフトの状態が始まるかまたは終わる文字列で、0 個以上の指示語 (もしあれば) で構成されます。各指示語は次のいずれか 1 つで構成されます。

- 1 つまたは複数の空白文字 (スペース、タブ、改行文字、垂直タブ、または用紙送り (form-feed) 文字)
- 通常文字 (% または空白文字は含まない)
- 変換指定

### 変換指定

各変換指定は % 文字、または %n\$ 文字シーケンスによって導入されます。これらの文字の後には、以下に示す (順番で) 文字列が続きます。

- 代入抑制文字 \* (任意)
- 最大フィールド幅を指定する 0 以外の 10 進数 (任意)
- 受け取るオブジェクトのサイズを表すサイズ修飾子、h、l (小文字のエル)、ll (小文字のエル 2 つ)、または L (任意)。対応する引数が `int` ではなく `short int` へのポインタである場合は、変換文字 d、i、および n の前に h を指定する必要があります。また、対応する引数が `long int` へのポインタである場合は、変換文字 d、i、および n の前に l (小文字のエル) を指定する必要があります。対応する引数が `long long int` へのポインタである場合は、変換文字 d、i、および n の前に ll (小文字のエル 2 つ) を指定する必要があります。同様に、対応する引数が `unsigned int` へのポインタではなく `unsigned short int` へのポインタである場合は、変換文字 o、u、および x の前に h を指定する必要があります。また、対応する引数が `unsigned long int` へのポインタである場合は、変換文字 o、u、および x の前に l (小文字のエル) を指定する必要があります。対応する引数が `unsigned long long int` へのポインタである場合は、変換文字 o、u、および x の前に ll (小文字のエル 2 つ) を指定する必要があります。対応する引数が `float` へのポインタではなく `double` へのポインタである場合は、変換文字 e、f、および g の前に l (小文字のエル) を指定する必要があります。または、対応する引数が `long double` へのポインタである場合は、変換文字 e、f、および g の前に L を指定する必要があります。最後に、対応する引数が文字形式へのポインタではなく `wchar_t` へのポインタである場合は、変換文字 c、s、および [ の前に l (小文字のエル) を指定する必要があります。h、l (小文字のエル)、ll (小文字のエル 2 つ)、および L が上記以外の変換文字の前に現われたとき、変換の結果は未定義となります。
- 適用される変換形式を指定する変換文字 (有効な変換文字については、後述の説明を参照してください)。

`fscanf()` 関数は、形式の各指示を順番に実行します。指示にエラーがあった場合、以下に詳しく記述しているように、関数はエラーを返します。エラーは、入力エラー (入力バイトが無効) またはマッチングエラー (入力が不適當) として記述されます。

1 つまたは複数の空白文字で構成される指示は、有効な入力を読み取られなくなるまで、あるいは空白文字でないために読み取れずに残る最初のバイトまで、入力を読み取って実行します。

通常文字の指示は次のように実行されます。次のバイトが入力から読み取られ、指示を含むバイトと比較されます。比較の結果、両者が同じでない場合、指示はエラーで終了し、違いのあった後続のバイトが読み取れずに残ります。

変換指定を表す指示は、一連のマッチング入力シーケンスを定義します (各変換文字については以下に記述)。変換指定は次の手順で実行されます。

変換指定に変換文字 `]`、`c`、`c`、または `n` が含まれる場合を除いて、空白文字の入力 (`isspace(3C)` で指定) はスキップされます。

変換指定に変換文字 `n` が含まれる場合を除いて、項目は入力から読み取られます。入力項目は、入力バイトの最も長いシーケンスとして (変換文字によっては文字またはバイト単位で測定される、指定された最大フィールド幅まで) 定義されます。入力項目幅はマッチングシーケンスの最初のサブシーケンスです。入力項目に続く最初のバイト (もしあれば) は、読み取れずに残ります。入力項目の長さが 0 の場合、変換指定の実行はエラーで終了します。この状態は、ファイルの終わり、コード化エラー、あるいはストリームからの入力を妨げる読み取りエラー (この場合は入力エラー) でない限り、マッチングのエラーです。

変換文字が `%` ではない場合の入力項目 (あるいは変換指定が `%n` の場合の入力バイト数) は、変換文字に適切な形式に変換されます。入力項目がマッチングシーケンスではない場合、変換指定はエラーで終了します。この状態はマッチングエラーです。代入抑制文字が `*` で表された場合を除いて、変換の結果は、以前に変換結果を受け取っていない `format` 引数に続く最初の引数によって、ポイントされたオブジェクトに出力されます (変換指定が `%` によって呼び出されていない場合)。変換指定が文字シーケンス `%n$` によって呼び出された場合は、`n` 番目の引数に出力されます。このオブジェクトが適切な形式ではない場合、または変換の結果が提供された空間に対応できない場合、動作結果は未定義です。

変換文字 以下に、有効な変換文字を示します。

- d 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- i 整数 (符号は任意) と一致します。書式は、`strtoul()` の `base` 引数に値 0 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- o 8 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 8 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。

## fscanf(3C)

- u 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- x 16 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 16 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- e,f,g 符号付き浮動小数点数に一致します。書式は `strtod(3C)` の変換対象となる文字コードの並びと同じです。サイズ修飾子を指定しない場合、対応する引数は `float` へのポインタである必要があります。
- `printf(3C)` ファミリの関数が、ANSI/IEEE Std 754: 1985 標準を指定できるように、無限および NaN (浮動小数点形式にコード化された 7858 シンボリックエンティティ) 用に文字列表現を生成した場合、`scanf()` ファミリの関数はそれらを入力として認識します。
- s 空白でないバイトシーケンスに一致します。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- l (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。それぞれの文字は、`mbrtowc(3C)` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- [ 複数の文字 (*scanset* という) からなる、空でない文字シーケンスと一致します。この場合、通常行われる先行する空白のスキップは抑制されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL バイトを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- l (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- 変換指定は、*format* 文字列のすべての後続の文字、つまり、対応する右角括弧 (]) まで (または、その括弧も含む) が入ります。括弧内の文字 (*scanlist*) は *scanset* を構成しています。左角括弧の次の文字がサーカンフ



レックス (^) の場合は、サーカンフレックスと直角括弧の間 *scanlist* に入っていないすべての文字が *scanset* を構成しています。変換指定が [ ] または [^] で始まる場合は、この直角括弧は *scanlist* 内に含まれており、次の直角括弧が指定の終了を示す直角括弧になります。*scanlist* に - があり、その - が最初の文字ではない場合、最初の文字が ^ で - が 2 番目の文字ではない場合、あるいは - が最後の文字ではない場合、一致する文字の範囲が表示されます。

c フィールド幅に指定された数 (フィールド幅が変換指定にない場合は 1) の文字シーケンスと一致します。対応する引数は、その文字シーケンスを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。NULL 文字は追加されません。この場合、通常行われる先行する空白のスキップは抑制されます。

l (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよび最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。NULL ワイド文字は追加されません。

p 対応する `printf(3C)` 関数の `%p` 変換によって出力されるシーケンスと同様のシーケンスと一致します。対応する引数は `void` へのポインタである必要があります。入力項目が、同一プログラムの実行中で以前に変換した値である場合は、ポインタは先にその値を変換したときと同様の比較を行います。そうでない場合は、`%p` 変換の動作は未定義です。

n 入力は使用 (変換) されません。対応する引数は、これまでに `scanf()` 関数を呼び出して入力ストリームから読み取ったバイト数を示す整数へのポインタである必要があります。変換指定 `%n` を実行しても、この関数の実行終了時に返される代入数は増加しません。

c 1c と同じです。

s ls と同じです。

% 単一の % と一致します。変換または代入は行われません。完全な変換指定をするには %% を指定してください。

変換指定が正しくない場合の動作は未定義です。

変換文字 E、G、および X はそれぞれ e、g、および x と同じ働きをします。

入力中にファイルの終端になると、変換は終了します。現在の変換指定 (`%n` を除く) に一致するバイト (先行空白を使用している場合はこれを除く) が読み取られる前にファイルが終わると、現在の変換指定の実行は入力エラーで終了します。それ以外の場合は、現在の変換指定の実行がマッチングエラーで終了しない限り、続く変換指定の実行 (もし、あれば) が入力エラーで終了します。

## fscanf(3C)

	<p>sscanf() の文字列の終端に到達するのは、fscanf() でファイルの終端に到達するのと同様です。</p> <p>入力が衝突して終了した場合、対抗する入力は入力中で読み取られないままになります。変換指定に一致しない場合、後続の空白(改行文字を含む)は読み取られません。リテラル一致および抑制される代入が成功したかどうかは、%n 変換指定によってのみ直接確認できます。</p> <p>fscanf() および scanf() 関数は、stream に関連するファイルの st_atime フィールドを更新用にマークすることができます。st_atime フィールドは、fgetc(3C), fgets(3C), fread(3C), fscanf(), getc(3C), getchar(3C), gets(3C), または scanf() の実行が成功したときに、ungetc(3C) への以前の呼び出しによって与えられていないデータを返す stream を使用して更新用にマークされます。</p>
戻り値	<p>正常終了の場合、これらの関数は一致と代入が成功した入力項目数を返します。一致が実行初期段階で失敗した場合は、0 が返される場合もあります。最初の一致が失敗する前、または最初の変換が行われる前に入力が終わると、EOF が返されます。ストリームが設定されているというエラー表示で読み取りエラーが発生した場合、EOF が返されエラーを表示する errno が設定されます。</p>
エラー	<p>scanf() 関数が失敗、および失敗する可能性がある場合は、fgetc(3C) または fgetwc(3C) を参照してください。</p> <p>さらに、以下の条件で fscanf() は失敗することがあります。</p> <p>EILSEQ            入力バイトシーケンスが有効な文字を形成していない。</p> <p>EINVAL            引数が不足している。</p>
使用法	<p>scanf() 関数を呼び出すアプリケーションに wint_t または wchar_t 型の形式のオブジェクトが存在する場合、これらのオブジェクトを定義する &lt;wchar.h&gt; ヘッダーも含む必要があります。</p>
使用例	<p><b>例 1 使用例 1</b></p> <pre>int i, n; float x; char name[50]; n = scanf("%d%f%s", &amp;i, &amp;x, name)</pre> <p>という呼び出しの場合、入力行は次のようになります。</p> <pre>25 54.32E-1 Hamster</pre> <p>この場合、n に値 3、i に値 25、x に値 5.432 がそれぞれ代入され、name には Hamster の文字列が入ります。</p> <p><b>例 2 使用例 2</b></p> <pre>int i; float x; char name[50]; (void) scanf("%2d%f*d %[0123456789]", &amp;i, &amp;x, name);</pre> <p>という呼び出しの場合、入力行は次のようになります。</p> <pre>56789 0123 56a72</pre>

## 例 2 使用例 2 (続き)

この場合、*i* に 56、*x* に 789.0 が代入され、0123 はスキップして、*name* には 56\0 が入ります。getchar(3C) への次の呼び出しでは a の文字を返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 fgetc(3C), fgets(3C), fgetwc(3C), fread(3C), isspace(3C), printf(3C), setlocale(3C), stdarg(3HEAD), strtod(3C), strtol(3C), strtoul(3C), wctomb(3C), ungetc(3C), attributes(5)

## getdate(3C)

名前	getdate – ユーザー形式の日付および時刻の変換
形式	<pre>#include &lt;time.h&gt;  struct tm *getdate(const char *string);  extern int getdate_err;</pre>
機能説明	<p>getdate() 関数は、<i>string</i> がポイントするユーザー定義の日付または時刻 (またはその両方) の書式指定を、tm 構造体に変換します。この構造体宣言は &lt;time.h&gt; ヘッダーにあります。</p> <p>ユーザーが作成したテンプレートは、入力文字列を構文解析して翻訳するときに使用されます。このテンプレートはテキストファイルで、DATEMSK 環境変数によって識別されます。テンプレートの各行は、strftime(3C) や strptime(3C) と同様の変換仕様で、受け入れ可能な日付と時刻 (またはその両方) の指定を表します。1902 年よりも前、または 2037 年よりも後の日付は正しくありません。入力指定と一致するテンプレートの最初の行は、内部時間形式への解釈および変換に使用されます。</p>
変換仕様	<p>以下の変換仕様がサポートされています。</p> <ul style="list-style-type: none"><li>%%        % と同じ</li><li>%a        省略形式の曜日名 (ロケール固有)</li><li>%A        完全形式の曜日名 (ロケール固有)</li><li>%b        省略形式の月名 (ロケール固有)</li><li>%B        完全形式の月名 (ロケール固有)</li><li>%c        日時の表現 (ロケール固有)</li><li>%C        世紀 (年を 100 で割った数値。1 から 99 までの 10 進数で表わされる。小数点以下は切り捨てられる)。1 桁の数値には、10 の位として 0 を指定する。standards(5) を参照してください。どの世紀を指定していても %y 指示子を指定せずに使用すると、現在の年を想定する。1902 年から 2037 年の間でのみ有効</li><li>%d        日 (1 から 31)。10 の位を示す 0 の指定は任意</li><li>%D        %m/%d/%y 形式で表した日付</li><li>%e        %d と同じ</li><li>%h        省略形式の月名 (ロケール固有)</li><li>%H        24 時間表示で表した時 (0 から 23)。10 の位を示す 0 の指定は任意</li><li>%I        12 時間表示で表した時 (1 から 12)。10 の位を示す 0 の指定は任意</li><li>%j        年の通算日 (1 から 366)。10 の位を示す 0 の指定は任意</li><li>%m        月を示す数値 (1 から 12)。10 の位を示す 0 の指定は任意</li><li>%M        分 (0 から 59)。10 の位を示す 0 の指定は任意</li></ul>

%n	空白
%p	午前または午後を表す値 (ロケール固有)
%r	12 時間表示形式で表した時刻。%p を伴う
%R	%H:%M 形式で表した時刻
%S	秒 (0 から 61)。10 の位を示す 0 の指定は任意。値の範囲が (00 から 59) で はなく (00 から 61) であるのは、うるう秒および二重うるう秒に対処する ため
%t	空白
%T	%H:%M:%S 形式で表した時刻
%U	その年の何週目かを表す数値 (0 から 53)。週は日曜日からはじまるとする。 10 の位を示す 0 の指定は任意
%w	曜日を表す番号 (0 から 6)。日曜日が 0
%W	その年の何週目かを表す数値 (0 から 53)。週は月曜日からはじまるとする。 10 の位を示す 0 の指定は任意
%x	ロケール固有の適切な日付の表現
%X	ロケール固有の適切な時刻の表現
%y	西暦年の下 2 桁。世紀が特に指定されていない場合、69 から 99 の範囲の 値は 20 世紀の西暦年 (1969 年から 1999 年まで) を表す。00 から 68 の範囲 の値は 21 世紀の西暦年 (2000 年から 2068 年まで) を表す
%Y	西暦年の完全表示 (たとえば 1993)
%Z	時間帯名 (時間帯がない場合には文字なし)
変換仕様の変更	上記の変換仕様に、変更を表す文字 E または O を付加すると、代替形式または代替指 定で値を得ることができます。希望した代替形式または代替指定が現ロケール中に存 在していなければ、変更を示す文字を指定しなかった場合の形式で値が得られます。
%Ec	適切な日付および時刻の代替形式 (ロケール固有)
%EC	代替表示形式として指定されている年号 (ロケール固有)
%Ex	日付の代替表示形式 (ロケール固有)
%EX	時刻の代替表示形式 (ロケール固有)
%Ey	代替表示形式として指定されている年号 (%EC) に対応した年 (ロケール固 有)
%EY	完全形式の代替年表示
%Od	日をロケール固有の代替数値記号で表す。10 の位を示す 0 の指定は任意
%Oe	%Od と同じ
%OH	24 時間表示の時をロケール固有の代替数値記号で表す

## getdate(3C)

%OI	12 時間表示の時をロケール固有の代替数値記号で表す
%Om	月をロケール固有の代替数値記号で表す
%OM	分をロケール固有の代替数値記号で表す
%OS	秒をロケール固有の代替数値記号で表す
%OU	その年の何週目かをロケール固有の代替数値記号で表す。週は日曜日から始まるとする
%Ow	曜日を示す値をロケール固有の代替数値記号で表す。日曜日を 0 とする
%OW	その年の何週目かをロケール固有の代替数値記号で表す。週は月曜日から始まるとする
%Oy	ロケール固有の代替表示の年 (%c に対応) の値を、ロケール固有の代替数値記号で表す

内部形式変換 以下の規則は入力指定を内部形式に変換する場合に適用されます。

- 曜日だけが指定されている場合、指定した曜日が現在の曜日ならば今日であると見なし、指定した曜日が現在の曜日よりも前であれば来週であると見なします。
- 月だけが指定されている場合、指定した月が現在月ならば今月であると見なし、指定した月が現在の月よりも前で年が指定されていない場合は来年であると見なします (日が指定されていない場合は、月の第 1 日目であると見なします)。
- 年だけが指定されている場合、返される `tm` 構造体のメンバー `tm_mon`、`tm_mday`、`tm_yday`、`tm_wday`、`tm_isdst` の値は指定されません。
- 世紀が指定されているが、その世紀の範囲内の年が指定されていない場合、現在の年はその世紀の範囲内であると見なします。
- 時間、分、秒が指定されていない場合、現在の時間、分、秒であると見なします。
- 日付が指定されていない場合、指定した時間が現在時間よりも大きければ今日であると見なし、指定した時間が現在の時間よりも小さければ明日であると見なします。

一般的な規則 通常の文字を使った変換仕様は、次の文字をバッファから検索することにより実行されます。検索した文字が変換仕様を構成する文字と異なっていると、変換仕様は誤りとなり、その文字以降は検索されません。

一連の変換仕様が `%n`、`%t`、空白、またはそれらの組み合わせで記述されている場合、空白でない文字が見つかるまで (その文字は読み込まれない)、またはすべての文字を検索し終わるまで、検索が行われます。

それ以外の変換仕様の場合には、次の変換仕様に一致する文字が見つかるまで、またはすべての文字を検索し終わるまで、検索が行われます。検索して読み込まれた文字群 (次の変換仕様に一致する文字は含まない) は、変換仕様に対応したロケール値と対応付けられます。一致するロケール値があれば、該当する `tm` 構造体メンバーの値がロケール情報に対応した値に設定されます。一致するロケール値がなければ、`getdate()` の実行は失敗し、文字検索は中止されます。

## getdate(3C)

月や曜日の名前、年号、代替数値記号には、大文字と小文字を任意に組み合わせることができます。また、setlocale(3C) を使って LC\_TIME カテゴリを設定することにより、日付や時刻の指定に特定の言語を使用するように指定できます。

戻り値 正常終了の場合、getdate() は、tm 構造体にポインタを返します。そうでない場合は NULL を返し、エラーを示す大域変数 getdate\_err を設定します。getdate\_err の内容は、getdate() 以降の呼び出しによって変更されます。

次に、getdate\_err の設定とその意味を示します。

- 1 DATEMSK 環境変数が NULL または未定義
- 2 読み取り用にテンプレートファイルを開くことができない
- 3 ファイルの状態情報を取得できない
- 4 テンプレートファイルが通常ファイルでない
- 5 テンプレートファイルの読み取り中にエラーを検出した
- 6 malloc() 関数の実行に失敗した (メモリー不足)
- 7 入力と一致する行がテンプレートに含まれていない
- 8 入力の指定が不正 (例:February 31)

使用法 getdate() は ctype(3C) に定義されているマクロを明示的に利用します。

使用例 例1 getdate() 関数の例

テンプレートの内容として、以下のものが考えられます。

```
%m
%A %B %d %Y, %H:%M:%S
%A
%B
%m/%d/%y %I %p
%d,%m,%Y %H:%M
at %A the %dst of %B in %Y
run job at %I %p,%B %dnd
%A den %d. %B %Y %H.%M Uhr
```

このテンプレートを使った有効な入力指定の例を示します。

```
getdate("10/1/87 4 PM")
getdate("Friday")
getdate("Friday September 19 1987, 10:30:30")
getdate("24,9,1986 10:30")
getdate("at monday the 1st of december in 1986")
getdate("run job at 3 PM, december 2nd")
```

LANG 環境変数が de (ドイツ語の環境) に設定されている場合は、以下の例が有効になります。

```
getdate("freitag den 10. oktober 1986 10.30 Uhr")
```

## getdate(3C)

### 例 1 getdate() 関数の例 (続き)

ローカルの日時の指定もサポートされています。次に、テンプレート内にローカルの日付指定を定義する方法を示します。

呼び出し	テンプレート内の行
getdate("11/27/86")	%m/%d/%y
getdate("27.11.86")	%d.%m.%y
getdate("86-11-27")	%y-%m-%d
getdate("Friday 12:00:00")	%A %H:%M:%S

次に、内部形式変換の規則の例を示します。現在の日付は Mon Sep 22 12:19:47 EDT 1986 であり、LANG 環境変数は設定されていないとします。

入力	テンプレート内の行	日付
Mon	%a	Mon Sep 22 12:19:48 EDT 1986
Sun	%a	Sun Sep 28 12:19:49 EDT 1986
Fri	%a	Fri Sep 26 12:19:49 EDT 1986
September	%B	Mon Sep 1 12:19:49 EDT 1986
January	%B	Thu Jan 1 12:19:49 EST 1987
December	%B	Mon Dec 1 12:19:49 EST 1986
Sep Mon	%b %a	Mon Sep 1 12:19:50 EDT 1986
Jan Fri	%b %a	Fri Jan 2 12:19:50 EST 1987
Dec Mon	%b %a	Mon Dec 1 12:19:50 EST 1986
Jan Wed 1989	%b %a %Y	Wed Jan 4 12:19:51 EST 1989
Fri 9	%a %H	Fri Sep 26 09:00:00 EDT 1986
Feb 10:30	%b %H:%S	Sun Feb 1 10:00:30 EST 1987
10:30	%H:%M	Tue Sep 23 10:30:00 EDT 1986
13:30	%H:%M	Mon Sep 22 13:30:00 EDT 1986

属性 次の属性については、attributes(5) のマニュアルページを参照してください。



getdate(3C)

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目

cftime(3C), mktime(3C), setlocale(3C), strftime(3C), strptime(3C),  
attributes(5), environ(5), standards(5)

## gettext(3C)

名前	gettext, dgettext, dcgettext, ngettext, dngettext, dcngettext, textdomain, bindtextdomain, bind_textdomain_codeset – メッセージ処理関数
形式	
Solaris および GNU 互換	<pre>#include &lt;libintl.h&gt; char *gettext(const char *msgid); char *dgettext(const char *domainname, const char *msgid); char *textdomain(const char *domainname); char *bindtextdomain(const char *domainname, const char *dirname); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcgettext(const char *domainname, const char *msgid, int category);</pre>
GNU 互換	<pre>#include &lt;libintl.h&gt; char *ngettext(const char *msgid1, const char *msgid2, unsigned long int n); char *dngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n); char *bind_textdomain_codeset(const char *domainname, const char *codeset); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n, int category);</pre>
機能説明	<p>gettext(), dgettext(), dcgettext() 関数は、特定のドメイン内において、また、現在のロケールにおいて、指定の <i>msgid</i> 引数に基づいたターゲット文字列を取得します。gettext(), dgettext(), dcgettext() から返される文字列の長さは、これらの関数が呼び出されるまでは未定です。<i>msgid</i> 引数は、NULL で終わる文字列です。</p> <p>ngettext(), dngettext(), および dcngettext() 関数はそれぞれ、gettext(), dgettext(), および dcgettext() と同等ですが、GNU 互換メッセージカタログだけを処理することができ、複数形の処理方法が異なります。メッセージ文字列を検索するとき、ngettext(), dngettext(), および dcngettext() 関数は <i>msgid1</i> 引数を検索鍵として使用し、<i>n</i> 引数で複数形を決定します。メッセージカタログが見つからない場合、<i>n</i> == 1 であれば <i>msgid1</i> が返され、そうでなければ <i>msgid2</i> が返されます。</p> <p>環境変数 NLSPATH (environ(5) を参照) が、最初に LC_MESSAGES カタログの位置として検索されます。現在のロケールの LC_MESSAGES カテゴリの設定によって、gettext() および dgettext() が文字列取得に使用するロケールが決まります。</p>

*category* 引数は `dcgettext()` の使用するロケールを決めます。NLSPATH が定義されておらず、現在のロケールが C ロケールの場合、`gettext()`、`gettext()`、`dcgettext()` は、渡されたメッセージ文字列をそのまま返します。C 以外のロケールでは、NLSPATH が定義されていない場合、あるいは、NLSPATH で指定したコンポーネントでメッセージカタログが見つからない場合、メッセージカタログの検索には、次の段落で説明するスキームが使用されます。

LANGUAGE 環境変数は、使用する GNU 互換メッセージカタログを決定するために調べられます。LANGUAGE の値はコロン (:) で区切られたロケール名のリストです。LANGUAGE が定義されている場合、リストに指定されている順番で各ロケールが検索されます。そして、GNU 互換メッセージカタログが見つかった場合、そのメッセージが返されます。GNU 互換メッセージが見つかったが、対応する *msgid* が見つからない場合、*msgid* 文字列が返されます。LANGUAGE が定義されていない場合、あるいは、LANGUAGE の処理中に Solaris メッセージカタログが見つかったが GNU 互換メッセージカタログが見つからない場合、*dirname/locale/category/domainname.mo* というパス名でメッセージカタログが検索されます。ここで、*dirname* は `bindtextdomain()` で指定されたディレクトリ、*locale* はロケール名、*category* は LC\_MESSAGES または LC\_XXX のどちらかです。LC\_MESSAGES は `gettext()`、`dgettext()`、`ngettext()`、または `dngettext()` を呼び出した場合に使用されます。LC\_XXX は `dcgettext()` または `dcngettext()` を呼び出した場合に使用され、XXX 部分は *category* 引数で指定したロケールカテゴリ名と同じになります。

`gettext()` と `ngettext()` の場合、使用されるドメインは、最後に行なった `textdomain()` の有効な呼び出しによって設定されます。`textdomain()` の有効な呼び出しが行われていない場合は、デフォルト時のドメイン (`messages`) が使用されます。

`dgettext()`、`dngettext()`、`dcngettext()`、および `dcgettext()` の場合、使用されるドメインは *domainname* 引数で指定されます。この *domainname* 引数は、構文と意味において、`textdomain()` の *domainname* 引数と同等ですが、このドメインの選択が `dgettext()`、`dcgettext()`、`dngettext()` および `dcngettext()` 関数の呼び出し中にのみ有効であるという点で異なります。

`textdomain()` 関数は、アクティブな LC\_MESSAGES ロケールカテゴリの現在のドメイン名の設定 または照会を行います。*domainname* 引数は、正当なファイル名に使用できる文字のみからなる、NULL で終わる文字列です。

*domainname* 引数は、システム内の、一意のドメイン名です。1つのシステム内に、同一ドメインのバージョンが複数存在する場合は、`bindtextdomain()` を使用して名前空間の衝突を回避することができます。`textdomain()` を呼び出さない場合には、デフォルト時のドメインが選択されます。最後に行なった `textdomain()` の有効な呼び出しによるドメインの設定は、このあとに行う `setlocale(3C)` および `gettext()` の呼び出しにおいても有効です。

*domainname* 引数は、現在アクティブな、LC\_MESSAGES ロケールに適用されます。

## gettext(3C)

*domainname* に NULL ポインタを設定して `textdomain()` を呼び出すことによって、ドメインの現在の状態に影響することなく、ドメインの現在の設定を照会することができます。*domainname* 引数に NULL 文字列を設定して `textdomain()` を呼び出すと、ドメインの設定は、デフォルト時のドメイン (`messages`) になります。

`bindtextdomain()` 関数は、メッセージドメインのパス部分 *domainname* と、*dirname* に入っている値を結合します。*domainname* が空文字列ではなく、まだ結合されていない場合には、`bindtextdomain()` は、*domainname* と *dirname* を結合します。

*domainname* が空文字列ではなく、すでに結合されている場合には、`bindtextdomain()` は、事前に行われていた結合部分を *dirname* に置き換えます。*dirname* 引数に使用できるのは、`gettext()`、`dgettext()`、または `dcgettext()` が呼び出されたときに解決される、絶対パス名または相対パス名です。*domainname* が NULL ポインタまたは空文字列の場合は、`bindtextdomain()` は NULL を返します。ユーザー定義のドメイン名は、`sys_` で始めることはできません。この文字列で始まるドメイン名は、システムで使用するために予約されています。

`bind_textdomain_codeset()` 関数を使用すると、ドメイン *domainname* のメッセージカタログ用の出力コードセットを指定できます。*codeset* 引数は、`iconv_open(3C)` 関数に対して使用可能な有効なコードセット名であるか、空ポインタである必要があります。*codeset* 引数が空ポインタの場合、`bind_textdomain_codeset()` はドメイン *domainname* 用に現在選択されているコードセットを返します。コードセットが選択されていない場合、`bind_textdomain_codeset()` は空ポインタを返します。`bind_textdomain_codeset()` 関数は複数回使用できます。同じ *domainname* 引数を指定して `bind_textdomain_codeset()` 関数を複数回使用した場合、先行する呼び出しで行われた設定は後続の呼び出しによって無効にされます。`bind_textdomain_codeset()` 関数は、選択されたコードセット名を含む文字列へのポインタを返します。この文字列は `bind_textdomain_codeset()` 関数に内部的に割り当てられているため、ユーザーが変更してはなりません。

### 戻り値

`gettext()`、`dgettext()`、および `dcgettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*msgid* 文字列を返します。

`ngettext()`、`dngettext()`、および `dcngettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*n* == 1 であれば、*msgid1* を返します。そうでなければ、*msgid2* を返します。

`gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` が返す文字列の個々のバイトには、空文字以外のあらゆる値が入ります。*msgid* が NULL ポインタである場合の戻り値は未定義です。返される文字列は、プログラムで修正してはならず、この後で `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` または `setlocale(3C)` を呼び出すことにより、無効になることがあります。`dgettext()`、`dcgettext()`、`dngettext()`、または `dcngettext()` に対する *domainname* 引数が NULL ポインタである場合、現在 `textdomain()` によってバインドされているドメインが使用されます。

## gettext(3C)

`textdomain()` の正常な戻り値は、ドメインの現在の設定が入っている文字列へのポインタです。`domainname` が NULL ポインタの場合は、`textdomain()` は現在のドメインが入っている文字列へのポインタを返します。`textdomain()` を先に呼び出していない場合に `domainname` を空文字列にすると、デフォルト時のドメイン名が返されます。デフォルト時のドメイン名は、`messages` です。`textdomain()` が失敗した場合、NULL ポインタが返されます。

`bindtextdomain()` からの戻り値は、`dirname` が入った NULL で終わる文字列です。`dirname` が NULL の場合は、`domainname` と結び付けられたディレクトリバインディングです。何も結合されていない場合には、デフォルトの戻り値は `/usr/lib/locale` になります。`domainname` が NULL ポインタまたは空文字列の場合、`bindtextdomain()` は何も行わず、NULL ポインタを返します。返された文字列を呼び出し側で変更してはなりません。`bindtextdomain()` が失敗した場合、NULL ポインタが返されます。

使用法 これらのルーチンでは、メッセージの長さに制限はありません。ただし、テキスト `domainname` は、TEXTDOMAINMAX (256) バイト以内に制限されています。

ロケール変更のために `setlocale(3C)` が呼び出されていない限り、マルチスレッドアプリケーションにおいて `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()`、`textdomain()`、および `bindtextdomain()` 関数を安全に使うことができます。

`gettext()`、`dgettext()`、`dcgettext()`、`textdomain()`、および `bindtextdomain()` 関数は Solaris メッセージカタログと GNU 互換メッセージカタログの両方を処理できます。`ngettext()`、`dngettext()`、`dcngettext()`、および `bind_textdomain_codeset()` 関数は GNU 互換メッセージカタログだけを処理できます。Solaris メッセージカタログと GNU 互換メッセージカタログの詳細については、`msgfmt(1)` のマニュアルページを参照してください。

ファイル

`/usr/lib/locale`

メッセージ・ドメイン・ファイルのデフォルトパス部分

`/usr/lib/locale/locale/LC_MESSAGES/domainname.mo`

言語 `locale` および `domainname` 用メッセージの入っているファイルのシステムデフォルト位置

`/usr/lib/locale/locale/LC_XXX/domainname.mo`

`LC_XXX` が `LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` である場合に、`dcgettext()` 呼び出し用の言語 `locale` および `domainname` メッセージの入っているファイルのシステムデフォルト位置

`dirname/locale/LC_MESSAGES/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の、ドメイン `domainname` およびパス部分 `dirname` 用メッセージが入っているファイルの位置

`dirname/locale/LC_XXX/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の `LC_XXX` が、`LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` のいずれかである場合 `dcgettext()` 呼び出し用のドメイン `domainname`、言語

## gettext(3C)

属性 *locale*、およびパス部分 *dirname* メッセージが入っているファイルの位置  
次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 `msgfmt(1)`, `xgettext(1)`, `iconv_open(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

名前 getwc – ストリームからワイド文字を読み込む

形式 

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

wint_t getwc(FILE *stream);
```

機能説明 getwc() 関数の機能は fgetwc(3C) 関数とほぼ同じです。異なる点は、マクロとして定義されている場合に *stream* を複数回評価することがあるので、複数回評価すると問題が起こる式を引数として指定してはならない、という点です。

戻り値 fgetwc(3C) の説明を参照してください。

エラー fgetwc(3C) の説明を参照してください。

使用法 この関数は、現在稼働中である一部のシステムに合わせるため、さらに将来的には ISO 標準に合わせるためのインタフェースとして提供されています。

getwc() はマクロとして定義されることがあるので、複数回評価すると問題が起こる *stream* 引数を正しく処理しない場合があります。特に getwc (\*f++) 指定は、期待どおりには動作しません。したがって、この関数ではなく fgetwc(3C) を使うことをお勧めします。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 fgetwc(3C), attributes(5)

## getwchar(3C)

名前 getwchar – stdin ストリームからワイド文字を読み込み

形式 

```
#include <wchar.h>

wint_t getwchar(void);
```

機能説明 getwchar() 関数は getwc(stdin) 関数と同じ機能です。

戻り値 fgetwc(3C) の説明を参照してください。

エラー fgetwc(3C) の説明を参照してください。

使用法 getwchar() が返す wint\_t の値を wchar\_t 型の変数に代入し、それを wint\_t のマクロ WEOF と比較しても、wchar\_t が符号なしと定義されているためにうまく比較できないことがあります。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 fgetwc(3C), getwc(3C), attributes(5)



名前 | getwidth – コードセット情報の取得

形式 | #include <euc.h>  
#include <getwidth.h>

```
void getwidth(eucwidth_t *ptr);
```

機能説明 | getwidth() 関数は、現在のロケール用の文字クラステーブルを読み込み、補助コードセットに関する情報を取得します。getwidth() は、この情報を構造体 eucwidth\_t に設定します。この構造体は <euc.h> に定義されており、以下の要素を含んでいます。

```
short int  _eucw1, _eucw2, _eucw3;
short int  _scrw1, _scrw2, _scrw3;
short int  _pcw;
char       _multibyte;
```

補助コードセット 1、2、および 3 に対するコードセット幅の値は、それぞれ \_eucw1、\_eucw2、および \_eucw3 に設定されています。補助コードセット 1、2 および 3 に対するスクリーン幅の値は、それぞれ \_scrw1、\_scrw2、および \_scrw3 に設定されています。

ワイド文字の幅は、\_pcw に設定されています。\_multibyte エントリは、複数バイト文字を使用する場合は 1 に、単一バイト文字だけを使用する場合は 0 に設定されます。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 | euclen(3C), setlocale(3C), attributes(5)

注意事項 | ロケール変更のために setlocale(3C) が呼び出されていない限り、マルチスレッドアプリケーションにおいて getwidth() 関数を安全に使うことができます。

getwidth() 関数は EUC でのみ動作します。

## getws(3C)

名前 getws, fgetws – ストリームからワイド文字列を取得する

形式

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

wchar_t *getws(wchar_t *ws);

#include <stdio.h>
#include <wchar.h>

wchar_t *fgetws(wchar_t *ws, int n, FILE *stream);
```

機能説明 getws() 関数は、標準入力ストリーム stdin から文字列を読み取り、それをワイド文字に変換し、ws が指す配列に書き込みます。この動作は、復帰改行文字が読み取られて変換され、ws に送られるまで、またはファイルの終端状態になるまで続けられます。ワイド文字列 ws は、ワイド文字コードの NULL 文字で終了します。

fgetws() 関数は、stream から文字を読み取り、それをワイド文字に変換し、ws が指す wchar\_t 配列に書き込みます。この動作は、n-1 文字まで読み取られるか、復帰改行文字が読み取られて変換され、ws に送られるまで、またはファイルの終端状態になるまで続けられます。ワイド文字列 ws は、ワイド文字コードの NULL 文字で終了します。

エラーが発生すると、ストリームに対するファイル指示子の戻り値が仲介になります。

fgetws() 関数は、stream に関連づけられているファイルの st\_atime フィールドに、更新のマークを付けます。fgetc(3C)、fgets(3C)、fgetwc(3C)、fgetws()、fread(3C)、fscanf(3C)、getc(3C)、getchar(3C)、gets(3C)、scanf(3C) の実行が最初に成功したときに、直前の scanf(3C) への呼び出しまたは scanf(3C) によって与えられないデータを返す stream を使用して、st\_atime フィールドに更新のマークが付けられます。

戻り値 getws() および fgetws() は、実行が成功すると ws を返します。ストリームがファイルの終端である場合、そのストリームについてファイル終端を示す指示子が設定され、fgetws() 関数が NULL ポインタを返します。読み取りエラーが発生した場合、ストリームに対してエラー指示子が設定され、fgetws() が NULL ポインタを返し、エラーを示す errno を設定します。

エラー fgetws() が失敗する状態については、fgetwc(3C) を参照してください。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 ferror(3C)、fgetwc(3C)、fread(3C)、getwc(3C)、putws(3C)、scanf(3C)、attributes(5)

名前 iconv - コード変換関数

形式 #include <iconv.h>

```
size_t iconv(iconv_t cd, const char **inbuf, size_t *inbytesleft, char
             **outbuf, size_t *outbytesleft);
```

機能説明 iconv() 関数は、*inbuf* で指定された配列内にある一連の文字のコードセットを変換します。具体的には、各文字を他のコードセットの対応する文字に変換し、結果を *outbuf* で指定された配列に書き出します。入力側と出力側のコードセットは、変換記述子 *cd* を返した *iconv\_open()* 呼び出し中に指定されていたものです。*inbuf* 引数は、入力バッファ中の先頭文字を指す変数を指します。*inbytesleft* は、変換対象となるバッファの終端までのバイト数を表す変数を指します。*outbuf* 引数は、出力バッファ内で使用可能な先頭バイトを指す変数を指します。*outbytesleft* は、出力バッファの終端までの使用可能なバイト数を表す変数を指します。

状態に依存するエンコーディングでは、変換記述子 *cd* は、*inbuf* が NULL ポインタである呼び出しまたは *inbuf* が NULL ポインタを指している呼び出しにより、初期のシフト状態に置かれます。このようにして *iconv()* が呼び出されたとき、*outbuf* が NULL ポインタでも NULL ポインタへのポインタでもない場合、*outbytesleft* が正の値を指していれば、*iconv()* はバイトシーケンスを出力バッファに書き込み、それにより出力バッファは初期のシフト状態に変更されます。リセットシーケンス全体を格納するのに必要なスペースが出力バッファ中になければ、*iconv()* は *errno* を E2BIG に設定して異常終了します。その後、*inbuf* が NULL ポインタでもなく NULL ポインタへのポインタでもない *iconv()* を呼び出すと、その時点の変換記述子に対して変換が行われます。

入力側の一連のバイトが、指定されたコードセット中の正当な文字を表していない場合には、最後に正しく変換された文字までで変換が終了します。入力バッファ中のデータが、文字やシフトシーケンスの途中で終わってしまっている場合には、最後に正しく変換されたバイトまでで変換が終了します。出力バッファが変換後のデータをすべて格納するのに十分な大きさでない場合、オーバーフローを起こすバイトの直前までで変換が終了します。*inbuf* がポイントする変数は、正しく変換された最後のバイトの直後のバイトを指すように更新されます。*inbytesleft* が示す値は、入力バッファ内に残されている未変換のバイト数を表すように減算されます。*outbuf* が指す変数は、変換して出力されたデータの最終バイトの次のバイトをポイントするように更新されます。*outbytesleft* が示す値は、出力バッファ中で使用可能な残りバイト数を表すように減算されます。状態に依存するエンコーディングのために、変換記述子は、最後に正しく変換されたバイトシーケンスの変換終了時に有効だったシフト状態を表すように更新されます。

正当ではあるが対応する文字が出力側コードセットにない、という文字を入力バッファ中に検出した場合、*iconv()* はシステムが定義したその文字を変換します。

戻り値 *iconv()* 関数は、引数で示された変数を更新して、どこまで変換が行われたかを表すようにします。また、出力側コードセットに該当する文字が存在しなかった変換の数を返します。入力バッファ中の文字列全体が変換されれば、*inbytesleft* が指す値は

## iconv(3C)

0 となります。前述した理由のいずれかにより変換が途中で停止した場合には、*inbytesleft* の値は 0 以外となり、状況を表す値が *errno* に設定されます。エラーが発生した場合、*iconv()* はエラーの状況を表す値を *errno* に設定して *(size\_t)-1* の値を返します。

エラー *iconv()* 関数は、以下のいずれかの場合に異常終了します。

EILSEQ	入力側のコードセットに属さないバイトが入力バッファ中に見つかり、変換を終了した
E2BIG	出力バッファ中に十分なスペースがないため、変換を終了した
EINVAL	入力バッファのデータが文字またはシフトシーケンスの途中で終わったため、変換を終了した

*iconv()* は、以下のいずれかの場合に異常終了することがあります。

EBADF	<i>cd</i> 引数が、正しく、しかもオープンされている変換記述子を表していない
-------	--

使用例 例 1 *iconv()* 関数の使用

次に *iconv()* 関数を使用する例を示します。

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <iconv.h>
#include <stdlib.h>

/*
 * For state-dependent encodings, changes the state of the conversion
 * descriptor to initial shift state. Also, outputs the byte sequence
 * to change the state to initial state.
 * This code is assuming the iconv call for initializing the state
 * won't fail due to lack of space in the output buffer.
 */
#define INIT_SHIFT_STATE(cd, fptr, ileft, tptr, oleft) \
{
    fptr = NULL; \
    ileft = 0; \
    tptr = to; \
    oleft = BUFSIZ; \
    (void) iconv(cd, &fptr, &ileft, &tptr, &oleft); \
    (void) fwrite(to, 1, BUFSIZ - oleft, stdout); \
}

int
main(int argc, char **argv)
{
    iconv_t cd;
    char    from[BUFSIZ], to[BUFSIZ];
    char    *from_code, *to_code;
    char    *tptr;
    const char *fptr;
    size_t  ileft, oleft, num, ret;
```

## 例 1 iconv() 関数の使用 (続き)

```

if (argc != 3) {
    (void) fprintf(stderr,
        "Usage: %s from_codeset to_codeset\\n", argv[0]);
    return (1);
}

from_code = argv[1];
to_code = argv[2];

cd = iconv_open((const char *)to_code, (const char *)from_code);
if (cd == (iconv_t)-1) {
    /*
     * iconv_open failed
     */
    (void) fprintf(stderr,
        "iconv_open(%s, %s) failed\\n", to_code, from_code);
    return (1);
}

ileft = 0;
while ((ileft +=
    (num = fread(from + ileft, 1, BUFSIZ - ileft, stdin))) > 0) {
    if (num == 0) {
        /*
         * Input buffer still contains incomplete character
         * or sequence.  However, no more input character.
         */

        /*
         * Initializes the conversion descriptor and outputs
         * the sequence to change the state to initial state.
         */
        INIT_SHIFT_STATE(cd, fptr, ileft, tptr, oleft);
        (void) iconv_close(cd);

        (void) fprintf(stderr, "Conversion error\\n");
        return (1);
    }

    fptr = from;
    for (;;) {
        tptr = to;
        oleft = BUFSIZ;

        ret = iconv(cd, &fptr, &ileft, &tptr, &oleft);
        if (ret != (size_t)-1) {
            /*
             * iconv succeeded
             */

            /*
             * Outputs converted characters

```

## iconv(3C)

### 例 1 iconv() 関数の使用 (続き)

```
        */
        (void) fwrite(to, 1, BUFSIZ - oleft, stdout);
        break;
    }

    /*
     * iconv failed
     */
    if (errno == EINVAL) {
        /*
         * Incomplete character or shift sequence
         */

        /*
         * Outputs converted characters
         */
        (void) fwrite(to, 1, BUFSIZ - oleft, stdout);
        /*
         * Copies remaining characters in input buffer
         * to the top of the input buffer.
         */
        (void) memmove(from, fptr, ileft);
        /*
         * Tries to fill input buffer from stdin
         */
        break;
    } else if (errno == E2BIG) {
        /*
         * Lack of space in output buffer
         */

        /*
         * Outputs converted characters
         */
        (void) fwrite(to, 1, BUFSIZ - oleft, stdout);
        /*
         * Tries to convert remaining characters in
         * input buffer with emptied output buffer
         */
        continue;
    } else if (errno == EILSEQ) {
        /*
         * Illegal character or shift sequence
         */

        /*
         * Outputs converted characters
         */
        (void) fwrite(to, 1, BUFSIZ - oleft, stdout);
        /*
         * Initializes the conversion descriptor and
         * outputs the sequence to change the state to
         * initial state.
         */
    }
}
```

## 例 1 iconv() 関数の使用 (続き)

```

        INIT_SHIFT_STATE(cd, fptr, ileft, tptr, oleft);
        (void) iconv_close(cd);

        (void) fprintf(stderr,
            "Illegal character or sequence\\n");
        return (1);
    } else if (errno == EBADF) {
        /*
         * Invalid conversion descriptor.
         * Actually, this shouldn't happen here.
         */
        (void) fprintf(stderr, "Conversion error\\n");
        return (1);
    } else {
        /*
         * This errno is not defined
         */
        (void) fprintf(stderr, "iconv error\\n");
        return (1);
    }
}

/*
 * Initializes the conversion descriptor and outputs
 * the sequence to change the state to initial state.
 */
INIT_SHIFT_STATE(cd, fptr, ileft, tptr, oleft);

(void) iconv_close(cd);
return (0);
}

```

ファイル /usr/lib/iconv/\*.so  
変換モジュール

/usr/lib/iconv/sparcv9/\*.so  
変換モジュール

/usr/lib/iconv/geniconvtbl/binarytables/\*.bt  
変換バイナリテーブル

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 geniconvtbl(1), iconv(1), iconv\_close(3C), iconv\_open(3C), geniconvtbl(4), attributes(5), iconv(5), iconv\_unicode(5)

## isalnum(3C)

名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に __XPG4_CHAR_CLASS__ を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任



		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の 10 進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の 16 進数の文字 ( [0-9]、[A-F]、または [a-f] ) について検査します。
標準準拠	<code>isxdigit()</code>	任意の 16 進数の文字 ( [0-9]、[A-F]、または [a-f] )、あるいは 16 進数を表現する ( 10 から 15 までの数字も含む ) 現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字 ( 文字または数字 ) について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィールド ( 標準空白文字 ) について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## isalnum(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。

isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)

名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に <code>__XPG4_CHAR_CLASS__</code> を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任

## isalpha(3C)

		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の10進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の16進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> ) について検査します。
標準準拠	<code>isxdigit()</code>	任意の16進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> )、あるいは16進数を表現する(10から15までの数字も含む)現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字(文字または数字)について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィード(標準空白文字)について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## isalpha(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。

isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

**戻り値** マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

**使用法** isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

**属性** 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

**関連項目** setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)

## isascii(3C)

名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に __XPG4_CHAR_CLASS__ を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任

		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字のASCII文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字のASCII文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の10進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の16進数の文字([0-9]、[A-F]、または[a-f])について検査します。
標準準拠	<code>isxdigit()</code>	任意の16進数の文字([0-9]、[A-F]、または[a-f])、あるいは16進数を表現する(10から15までの数字も含む)現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字(文字または数字)について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィールド(標準空白文字)について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## isascii(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。

isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)



名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に <code>__XPG4_CHAR_CLASS__</code> を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任

## iscntrl(3C)

		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の 10 進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の 16 進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> ) について検査します。
標準準拠	<code>isxdigit()</code>	任意の 16 進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> )、あるいは 16 進数を表現する (10 から 15 までの数字も含む) 現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字 (文字または数字) について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィード (標準空白文字) について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## iscntrl(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。

isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)

## isdigit(3C)

名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に __XPG4_CHAR_CLASS__ を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任

		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字のASCII文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字のASCII文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の10進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の16進数の文字([0-9]、[A-F]、または[a-f])について検査します。
標準準拠	<code>isxdigit()</code>	任意の16進数の文字([0-9]、[A-F]、または[a-f])、あるいは16進数を表現する(10から15までの数字も含む)現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字(文字または数字)について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィールド(標準空白文字)について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## isdigit(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。

isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

isenglish(3C)

iswgraph( <i>wc</i> )	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <b>graph</b> クラスの文字を表すワイド文字であるかどうかを検査します。
iswcntrl( <i>wc</i> )	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <b>cntrl</b> クラスの文字を表すワイド文字であるかどうかを検査します。
iswascii( <i>wc</i> )	<i>wc</i> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
isphonogram( <i>wc</i> )	<i>wc</i> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
isideogram( <i>wc</i> )	<i>wc</i> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
isenglish( <i>wc</i> )	<i>wc</i> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
isnumber( <i>wc</i> )	<i>wc</i> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
isspecial( <i>wc</i> )	<i>wc</i> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`



名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に <code>__XPG4_CHAR_CLASS__</code> を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任

## isgraph(3C)

		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の 10 進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の 16 進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> ) について検査します。
標準準拠	<code>isxdigit()</code>	任意の 16 進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> )、あるいは 16 進数を表現する (10 から 15 までの数字も含む) 現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字 (文字または数字) について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィード (標準空白文字) について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## isgraph(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。

isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)

## isideogram(3C)

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	<pre>#include &lt;wchar.h&gt;  int <b>iswalpha</b>(wint_t wc);</pre>
機能説明	<p>これらの関数は、<i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。</p> <p>すべての場合において、<i>wc</i> は <code>wint_t</code> 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの <code>WEOF</code> に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。</p> <p><code>iswalpha(wc)</code>                    プログラムの現在のロケールにおいて、<i>wc</i> が、<code>alpha</code> クラスの文字を表すワイド文字であるかどうかを検査します。</p> <p><code>iswupper(wc)</code>                    プログラムの現在のロケールにおいて、<i>wc</i> が、<code>upper</code> クラスの文字を表すワイド文字であるかどうかを検査します。</p> <p><code>iswlower(wc)</code>                    プログラムの現在のロケールにおいて、<i>wc</i> が、<code>lower</code> クラスの文字を表すワイド文字であるかどうかを検査します。</p> <p><code>iswdigit(wc)</code>                    プログラムの現在のロケールにおいて、<i>wc</i> が、<code>digit</code> クラスの文字を表すワイド文字であるかどうかを検査します。</p> <p><code>iswxdigit(wc)</code>                    プログラムの現在のロケールにおいて、<i>wc</i> が、<code>xdigit</code> クラスの文字を表すワイド文字であるかどうかを検査します。</p> <p><code>iswalnum(wc)</code>                    プログラムの現在のロケールにおいて、<i>wc</i> が、<code>alpha</code> クラスまたは <code>digit</code> クラスの文字を表すワイド文字であるかどうかを検査します。</p> <p><code>iswspace(wc)</code>                    プログラムの現在のロケールにおいて、<i>wc</i> が、<code>space</code> クラスの文字を表すワイド文字であるかどうかを検査します。</p> <p><code>iswpunct(wc)</code>                    プログラムの現在のロケールにおいて、<i>wc</i> が、<code>punct</code> クラスの文字を表すワイド文字であるかどうかを検査します。</p> <p><code>iswprint(wc)</code>                    プログラムの現在のロケールにおいて、<i>wc</i> が、<code>print</code> クラスの文字を表すワイド文字であるかどうかを検査します。</p>

## isideogram(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<i>wc</i> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

## islower(3C)

名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に __XPG4_CHAR_CLASS__ を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任

		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字のASCII文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字のASCII文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の10進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の16進数の文字([0-9]、[A-F]、または[a-f])について検査します。
標準準拠	<code>isxdigit()</code>	任意の16進数の文字([0-9]、[A-F]、または[a-f])、あるいは16進数を表現する(10から15までの数字も含む)現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字(文字または数字)について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィールド(標準空白文字)について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## islower(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。

isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)



名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## isnumber(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<i>wc</i> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。  iswalpha( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。  iswupper( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。  iswlower( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。  iswdigit( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。  iswxdigit( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。  iswalnum( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。  iswspace( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。  iswpunct( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。  iswprint( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## isphonogram(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<i>wc</i> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に <code>__XPG4_CHAR_CLASS__</code> を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任

## isprint(3C)

		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の 10 進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の 16 進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> ) について検査します。
標準準拠	<code>isxdigit()</code>	任意の 16 進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> )、あるいは 16 進数を表現する (10 から 15 までの数字も含む) 現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字 (文字または数字) について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィード (標準空白文字) について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## isprint(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。

isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)

## ispunct(3C)

名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に __XPG4_CHAR_CLASS__ を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任



		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の 10 進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の 16 進数の文字 ([0-9]、[A-F]、または [a-f]) について検査します。
標準準拠	<code>isxdigit()</code>	任意の 16 進数の文字 ([0-9]、[A-F]、または [a-f])、あるいは 16 進数を表現する (10 から 15 までの数字も含む) 現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字 (文字または数字) について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィールド (標準空白文字) について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## ispunct(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。

isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)

名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に <code>__XPG4_CHAR_CLASS__</code> を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任

## isspace(3C)

		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の 10 進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の 16 進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> ) について検査します。
標準準拠	<code>isxdigit()</code>	任意の 16 進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> )、あるいは 16 進数を表現する (10 から 15 までの数字も含む) 現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字 (文字または数字) について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィード (標準空白文字) について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

isspace(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。  
 isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)

## isspecial(3C)

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

isspecial(3C)

iswgraph( <i>wc</i> )	プログラムの現在のロケールにおいて、 <i>wc</i> が、graph クラスの文字を表すワイド文字であるかどうかを検査します。
iswcntrl( <i>wc</i> )	プログラムの現在のロケールにおいて、 <i>wc</i> が、cntrl クラスの文字を表すワイド文字であるかどうかを検査します。
iswascii( <i>wc</i> )	<i>wc</i> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
isphonogram( <i>wc</i> )	<i>wc</i> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
isideogram( <i>wc</i> )	<i>wc</i> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
isenglish( <i>wc</i> )	<i>wc</i> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
isnumber( <i>wc</i> )	<i>wc</i> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
isspecial( <i>wc</i> )	<i>wc</i> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

## isupper(3C)

名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に __XPG4_CHAR_CLASS__ を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任



		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字のASCII文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字のASCII文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の10進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の16進数の文字([0-9]、[A-F]、または[a-f])について検査します。
標準準拠	<code>isxdigit()</code>	任意の16進数の文字([0-9]、[A-F]、または[a-f])、あるいは16進数を表現する(10から15までの数字も含む)現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字(文字または数字)について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィールド(標準空白文字)について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## isupper(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。

isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。  iswalpha( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。  iswupper( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。  iswlower( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。  iswdigit( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。  iswxdigit( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。  iswalnum( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。  iswspace( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。  iswpunct( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。  iswprint( <i>wc</i> )                    プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswalnum(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<code>wc</code> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表音文字 を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表意文字 を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ英語のアルファベット を表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ特殊文字 を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswalpha(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<i>wc</i> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswascii(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<i>wc</i> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`



名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswcntrl(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<code>wc</code> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

名前 iswctype – 指定された文字のクラスの検査

形式 #include <wchar.h>

```
int iswctype(wint_t wc, wctype_t charclass);
```

機能説明 iswctype() 関数は、ワイド文字 *wc* の文字クラスが *charclass* で示すクラスに等しいかどうかを検査し、TRUE または FALSE を返します。iswctype() 関数は、現在のロケールでの正しい文字コードに対応したワイド文字コード および WEOF に対して定義されています。引数 *wc* が関数のドメイン中に存在しない場合、実行結果は未定義です。*charclass* の値が正しくない場合、すなわち wctype(3C) を呼び出して得られた値でなかったり、カテゴリ LC\_CTYPE に影響を与えるような setlocale(3C) の呼び出しが後で発生して *charclass* が不当になってしまった場合、実行の結果は不確定です。

戻り値 文字クラスの検査結果が FALSE の場合は 0 が、TRUE の場合には 0 以外の値が返されます。

使用法 標準文字クラス名として予約されている 12 の文字列があります。

"alnum"	"alpha"	"blank"
"cntrl"	"digit"	"graph"
"lower"	"print"	"punct"
"space"	"upper"	"xdigit"

以下の表で、左側の関数と右側の関数は 等価です。

iswalnum( <i>wc</i> )	iswctype( <i>wc</i> , wctype("alnum"))
iswalpha( <i>wc</i> )	iswctype( <i>wc</i> , wctype("alpha"))
iswcntrl( <i>wc</i> )	iswctype( <i>wc</i> , wctype("cntrl"))
iswdigit( <i>wc</i> )	iswctype( <i>wc</i> , wctype("digit"))
iswgraph( <i>wc</i> )	iswctype( <i>wc</i> , wctype("graph"))
iswlower( <i>wc</i> )	iswctype( <i>wc</i> , wctype("lower"))
iswprint( <i>wc</i> )	iswctype( <i>wc</i> , wctype("print"))
iswpunct( <i>wc</i> )	iswctype( <i>wc</i> , wctype("punct"))
iswspace( <i>wc</i> )	iswctype( <i>wc</i> , wctype("space"))
iswupper( <i>wc</i> )	iswctype( <i>wc</i> , wctype("upper"))
iswxdigit( <i>wc</i> )	iswctype( <i>wc</i> , wctype("xdigit"))

## iswctype(3C)

なお、次の関数と等価な `isw*( )` 関数はありません。

```
iswctype(wc, wctype("blank"))
```

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 `iswalphabet(3C)`, `setlocale(3C)`, `wctype(3C)`, `attributes(5)`, `environ(5)`

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswdigit(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<code>wc</code> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表音文字 を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表意文字 を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ英語のアルファベット を表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ特殊文字 を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswgraph(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<code>wc</code> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`



名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswlower(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<i>wc</i> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswprint(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<code>wc</code> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswpunct(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<i>wc</i> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswspace(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<code>wc</code> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`



名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswupper(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <i>wc</i> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<i>wc</i> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<i>wc</i> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

名前	iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – ワイド文字の文字分類用関数
形式	#include <wchar.h>  int <b>iswalpha</b> (wint_t <i>wc</i> );
機能説明	これらの関数は、 <i>wc</i> が、現在のロケールの LC_CTYPE カテゴリに定義されている、各クラスの文字を表すワイド文字であるかどうかを検査します。  すべての場合において、 <i>wc</i> は wint_t 型であり、その値は、現在のロケールで有効な文字に対応するワイド文字であるか、または、マクロの WEOF に等しい値であることが必要です。引数に他の値がついている場合には、動作は未定義です。
	iswalpha( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスの文字を表すワイド文字であるかどうかを検査します。
	iswupper( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、upper クラスの文字を表すワイド文字であるかどうかを検査します。
	iswlower( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、lower クラスの文字を表すワイド文字であるかどうかを検査します。
	iswdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswxdigit( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、xdigit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswalnum( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、alpha クラスまたは digit クラスの文字を表すワイド文字であるかどうかを検査します。
	iswspace( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、space クラスの文字を表すワイド文字であるかどうかを検査します。
	iswpunct( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、punct クラスの文字を表すワイド文字であるかどうかを検査します。
	iswprint( <i>wc</i> )                      プログラムの現在のロケールにおいて、 <i>wc</i> が、print クラスの文字を表すワイド文字であるかどうかを検査します。

## iswxdigit(3C)

<code>iswgraph(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>graph</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswcntrl(wc)</code>	プログラムの現在のロケールにおいて、 <code>wc</code> が、 <code>cntrl</code> クラスの文字を表すワイド文字であるかどうかを検査します。
<code>iswascii(wc)</code>	<code>wc</code> が、ASCII 文字を表すワイド文字であるかどうかを検査します。
<code>isphonogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表音文字を表すワイド文字であるかどうかを検査します。
<code>isideogram(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ表意文字を表すワイド文字であるかどうかを検査します。
<code>isenglish(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ英語のアルファベットを表すワイド文字であるかどうかを検査します。
<code>isnumber(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ数字 [0-9] を表すワイド文字であるかどうかを検査します。
<code>isspecial(wc)</code>	<code>wc</code> が、ASCII 文字を含んだ特殊文字を表すワイド文字であるかどうかを検査します。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `localedef(1)`, `setlocale(3C)`, `stdio(3C)`, `ascii(5)`, `attributes(5)`

名前	ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – 文字処理
形式	<pre>#include &lt;ctype.h&gt;  int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre>
機能説明	<p>これらのマクロは文字 (char) コードを符号化した整数値を分類します。各マクロは真の場合には 0 以外を返し、偽の場合には 0 を返します。isascii() を除いて、これらのマクロの動作は現在のロケールに左右されます (setlocale(3C) を参照)。この動作を修正するには、setlocale() における LC_TYPE カテゴリ、すなわち、setlocale(LC_CTYPE, newlocale) を変更します。C ロケールまたは文字型情報が定義されていないロケールにおいて、文字は US-ASCII 7-bit コード化文字集合の規則にしたがって分類されています。</p> <p>isascii() マクロはすべての整数型値に関して定義されています。残りのマクロは、引数が int、その値は unsigned char または EOF として表現でき、&lt;stdio.h&gt; ヘッダーによって定義されていて、ファイルの終わりを表している場合に限り、定義されます。</p> <p>以下に定義したマクロにはすべて対応する関数があります。この関数書式を取り込む場合には、マクロ名を未定義にする必要があります (たとえば、#undef isdigit)。</p> <p>「デフォルト」と「標準準拠」で記述しているマクロでは、標準準拠の動作は標準準拠のアプリケーション (standards(5) 参照)、および &lt;ctype.h&gt; を含む前に <code>__XPG4_CHAR_CLASS__</code> を定義するアプリケーションで提供されます。</p>
デフォルト	isalpha() isupper() または islower() が真である任意の文字について検査します。
標準準拠	isalpha() isupper() または islower() が真である任意の文字、あるいは iscntrl()、isdigit()、ispunct()、isspace() のいずれも真でない現在のロケールの定義済み文字集合の 1 つである任

## isxdigit(3C)

		意の文字について検査します。C ロケールでは、 <code>isupper()</code> または <code>islower()</code> が真である文字に対してのみ、 <code>isalpha()</code> は真を返します。
	<code>isupper()</code>	任意の大文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>islower()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>isupper()</code> は大文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>islower()</code>	任意の小文字、または <code>iscntrl()</code> 、 <code>isdigit()</code> 、 <code>ispunct()</code> 、 <code>isspace()</code> 、 <code>isupper()</code> のいずれも真でない現在のロケールの定義済み文字集合の1つである任意の文字について検査します。C ロケールでは、 <code>islower()</code> は小文字の ASCII 文字として定義された文字についてだけ真を返します。
	<code>isdigit()</code>	任意の 10 進数の文字について検査します。
デフォルト	<code>isxdigit()</code>	任意の 16 進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> ) について検査します。
標準準拠	<code>isxdigit()</code>	任意の 16 進数の文字 ( <code>[0-9]</code> 、 <code>[A-F]</code> 、または <code>[a-f]</code> )、あるいは 16 進数を表現する (10 から 15 までの数字も含む) 現在のロケールの定義済み文字集合について検査します。C ロケールでは次の文字だけが含まれます。0123456789ABCDEFabcdef
	<code>isalnum()</code>	<code>isalpha()</code> または <code>isdigit()</code> が真である任意の文字 (文字または数字) について検査します。
	<code>isspace()</code>	空白文字、タブ、キャリッジリターン、復帰改行、垂直タブまたはフォームフィード (標準空白文字) について、あるいは <code>isalnum()</code> が偽である現在のロケールの定義済み文字集合の1つについて検査します。C ロケールにおいて、 <code>isspace()</code> は標準空白文字についてだけ真を返します。
	<code>ispunct()</code>	空白文字でなく、 <code>isalnum()</code> または <code>iscntrl()</code> が真でもない印刷文字について検査します。
デフォルト	<code>isprint()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> と空白文字が真である任意の文字について検査します。
標準準拠	<code>isprint()</code>	<code>iscntrl()</code> が偽であり、 <code>isalnum()</code> 、 <code>isgraph()</code> 、 <code>ispunct()</code> 、空白文字、現在のロケールで定義済みの <code>print</code> クラスの文字が真である任意の文字について検査します。
デフォルト	<code>isgraph()</code>	<code>ispunct()</code> 、 <code>isupper()</code> 、 <code>islower()</code> 、 <code>isdigit()</code> が真である任意の文字について検査します。
標準準拠	<code>isgraph()</code>	<code>isalnum()</code> 、 <code>ispunct()</code> が真である任意の文字、または、空白文字でなく、 <code>iscntrl()</code> が真でもない現在のロケールで定義済みの <code>graph</code> クラスの文字が真である任意の文字について検査します。

## isxdigit(3C)

iscntrl() 文字集合に定義されている「制御文字」について検査します。

isascii() 0 から 0177 までのコードを含む ASCII 文字を検査します。

次のマクロは Solaris と XPG4 とで異なる動作をします。

戻り値 マクロを処理する任意の文字への引数が関数の定義域にない場合、出力結果は未定義になります。そうでない場合、マクロまたは関数は、真のときには 0 以外を返し、偽のときには 0 を返します。

使用法 isdigit()、isxdigit()、islower()、isupper()、isalpha()、isalnum()、isspace()、iscntrl()、ispunct()、isprint()、isgraph()、または isascii() マクロは、setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッド上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), stdio(3C), ascii(5), environ(5), standards(5)

## mblen(3C)

名前 mblen – 文字のバイト数を取得する

形式 

```
#include <stdlib.h>

int mblen(const char *s, size_t n);
```

機能説明  $s$  が NULL ポインタでないとき、`mblen()` は  $s$  が示す文字のバイト数を調べます。以下の処理と同じ結果になります。

```
mbtowc((wchar_t *)0, s, n);
```

$s$  に NULL ポインタを指定して呼び出すと、`mblen()` は 0 を返します。この関数の動作は、現在のロケールの LC\_CTYPE カテゴリによって影響されます。

戻り値  $s$  が NULL ポインタの場合、`mblen()` は 0 を返します。 $s$  が NULL ポインタではない場合、`mblen()` は  $s$  が NULL バイトを指していれば 0 を返し、後続の  $n$  個以下のバイトが正しい文字を構成していればそのバイト数を、正しい文字を構成していなければ -1 を返して `errno` にエラーを示す値を設定します。どのような場合でも、返される値は  $n$  や `MB_CUR_MAX` マクロの値を超えることはありません。

エラー 以下の場合に `mblen()` 関数は異常終了します。

EILSEQ 不当な文字シーケンスが検出された

使用法 ロケール変更のために `setlocale(3C)` が呼び出されていないかぎり、マルチスレッドのアプリケーションにおいて `mblen()` 関数を安全に使用できます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `mbstowcs(3C)`, `mbtowc(3C)`, `setlocale(3C)`, `wcstombs(3C)`, `wctomb(3C)`, `attributes(5)`



名前 mbstowcs – 文字列をワイド文字列に変換

形式 #include <stdlib.h>

```
size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
```

機能説明 mbstowcs() 関数は、s が示す配列の文字シーケンスを、対応するワイド文字コードのシーケンスに変換し、これらのコードを n 個を超えない範囲で、pwcs が示す配列に格納します。NULL バイト (ワイド文字コード型の 0 に変換される) に続く文字は、検査または変換されません。それぞれの文字は mbtowc(3C) への呼び出しによる変換と同じように変換されます。

pwcs によって示される配列の n 個を超えない要素が変更されます。コピーが、オーバーラップするオブジェクト間で行われた場合には、動作は定義されません。

この関数の動作は、現在のロケールの LC\_CTYPE カテゴリに影響を受けます。pwcs が NULL ポインタである場合、mbstowcs() は、n の値にかかわらず、配列全体を変換するために必要な長さを返します。格納される値はありません。

戻り値 無効な文字が検出されると、mbstowcs() は (size\_t)-1 を返し、エラー表示のために errno を設定する場合があります。それ以外の場合には、mbstowcs() は、末尾の 0 コード (ある場合) を数えずに、変更された配列要素の数 (または pwcs が NULL の場合は必要な配列要素の数) を返します。戻り値が n の場合、配列の末尾に 0 は付加されません。

エラー mbstowcs() 関数は次の場合に失敗することがあります。

EILSEC 無効なバイトシーケンスが検出された

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
CSI	対応済み
インタフェースの安定性	標準
MT レベル	MT-Safe

関連項目 mblen(3C), mbtowc(3C), setlocale(3C), wcstombs(3C), wctomb(3C), attributes(5), standards(5)

## mbtowc(3C)

名前 mbtowc – 複数バイト文字をワイド文字コードに変換

形式 #include <stdlib.h>

```
int mbtowc(wchar_t *pwc, const char *s, size_t n);
```

機能説明 *s* が NULL ポインタでないとき、mbtowc() は *s* が示す文字のバイト数を判断します。さらに、その文字に対応する wchar\_t 型の値のワイド文字コードを調べます。NULL バイトに対応するワイド文字コードの値は 0 となります。文字が正しいものであり、さらに *pwc* が NULL ポインタでなければ、mbtowc() は対応するワイド文字コードを *pwc* が示すオブジェクト中に書き出します。

*s* に NULL ポインタを指定して呼び出すと、mbtowc() は 0 を返します。この関数の動作は、現在のロケールの LC\_CTYPE カテゴリによって影響されます。*s* が示す配列のうち、最大 *n* バイトが調べられます。

戻り値 *s* が NULL ポインタの場合、mbtowc() は 0 を返します。*s* が NULL ポインタではない場合、mbtowc() は *s* が NULL バイトを指していれば 0 を返し、後続の *n* 個以下のバイトが正しい文字を構成していればそのバイト数を、正しい文字を構成していなければ -1 を返して errno にエラーを示す値を設定します。

どのような場合でも、返される値は *n* や MB\_CUR\_MAX マクロの値を超えることはありません。

エラー 以下の場合に mbtowc() 関数は異常終了します。

EILSEQ 不当な文字シーケンスが検出された

使用法 ロケール変更のために mbtowc() setlocale(3C) を呼び出していないかぎり、マルチスレッドのアプリケーションにおいて、mbtowc() 関数を安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 mblen(3C), mbstowcs(3C), setlocale(3C), wcstombs(3C), wctomb(3C), attributes(5)

名前	gettext, dgettext, dcgettext, ngettext, dngettext, dcngettext, textdomain, bindtextdomain, bind_textdomain_codeset – メッセージ処理関数
形式	
Solaris および GNU 互換	<pre>#include &lt;libintl.h&gt; char *gettext(const char *msgid); char *dgettext(const char *domainname, const char *msgid); char *textdomain(const char *domainname); char *bindtextdomain(const char *domainname, const char *dirname); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcgettext(const char *domainname, const char *msgid, int category);</pre>
GNU 互換	<pre>#include &lt;libintl.h&gt; char *ngettext(const char *msgid1, const char *msgid2, unsigned long int n); char *dngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n); char *bind_textdomain_codeset(const char *domainname, const char *codeset); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n, int category);</pre>
機能説明	<p>gettext(), dgettext(), dcgettext() 関数は、特定のドメイン内において、また、現在のロケールにおいて、指定の <i>msgid</i> 引数に基づいたターゲット文字列を取得します。gettext(), dgettext(), dcgettext() から返される文字列の長さは、これらの関数が呼び出されるまでは未定です。<i>msgid</i> 引数は、NULL で終わる文字列です。</p> <p>ngettext(), dngettext(), および dcngettext() 関数はそれぞれ、gettext(), dgettext(), および dcgettext() と同等ですが、GNU 互換メッセージカタログだけを処理することができ、複数形の処理方法が異なります。メッセージ文字列を検索するとき、ngettext(), dngettext(), および dcngettext() 関数は <i>msgid1</i> 引数を検索鍵として使用し、<i>n</i> 引数で複数形を決定します。メッセージカタログが見つからない場合、<i>n</i> == 1 であれば <i>msgid1</i> が返され、そうでなければ <i>msgid2</i> が返されます。</p> <p>環境変数 NLSPATH (environ(5) を参照) が、最初に LC_MESSAGES カタログの位置として検索されます。現在のロケールの LC_MESSAGES カテゴリの設定によって、gettext() および dgettext() が文字列取得に使用するロケールが決まります。</p>

*category* 引数は `dcgettext()` の使用するロケールを決めます。NLSPATH が定義されておらず、現在のロケールが C ロケールの場合、`gettext()`、`gettext()`、`dcgettext()` は、渡されたメッセージ文字列をそのまま返します。C 以外のロケールでは、NLSPATH が定義されていない場合、あるいは、NLSPATH で指定したコンポーネントでメッセージカタログが見つからない場合、メッセージカタログの検索には、次の段落で説明するスキームが使用されます。

LANGUAGE 環境変数は、使用する GNU 互換メッセージカタログを決定するために調べられます。LANGUAGE の値はコロン (:) で区切られたロケール名のリストです。LANGUAGE が定義されている場合、リストに指定されている順番で各ロケールが検索されます。そして、GNU 互換メッセージカタログが見つかった場合、そのメッセージが返されます。GNU 互換メッセージが見つかったが、対応する *msgid* が見つからない場合、*msgid* 文字列が返されます。LANGUAGE が定義されていない場合、あるいは、LANGUAGE の処理中に Solaris メッセージカタログが見つかったが GNU 互換メッセージカタログが見つからない場合、*dirname/locale/category/domainname.mo* というパス名でメッセージカタログが検索されます。ここで、*dirname* は `bindtextdomain()` で指定されたディレクトリ、*locale* はロケール名、*category* は LC\_MESSAGES または LC\_XXX のどちらかです。LC\_MESSAGES は `gettext()`、`dgettext()`、`ngettext()`、または `dngettext()` を呼び出した場合に使用されます。LC\_XXX は `dcgettext()` または `dcngettext()` を呼び出した場合に使用され、XXX 部分は *category* 引数で指定したロケールカテゴリ名と同じになります。

`gettext()` と `ngettext()` の場合、使用されるドメインは、最後に行なった `textdomain()` の有効な呼び出しによって設定されます。`textdomain()` の有効な呼び出しが行われていない場合は、デフォルト時のドメイン (`messages`) が使用されます。

`dgettext()`、`dngettext()`、`dcngettext()`、および `dcgettext()` の場合、使用されるドメインは *domainname* 引数で指定されます。この *domainname* 引数は、構文と意味において、`textdomain()` の *domainname* 引数と同等ですが、このドメインの選択が `dgettext()`、`dcgettext()`、`dngettext()` および `dcngettext()` 関数の呼び出し中にのみ有効であるという点で異なります。

`textdomain()` 関数は、アクティブな LC\_MESSAGES ロケールカテゴリの現在のドメイン名の設定 または照会を行います。*domainname* 引数は、正当なファイル名に使用できる文字のみからなる、NULL で終わる文字列です。

*domainname* 引数は、システム内の、一意のドメイン名です。1つのシステム内に、同一ドメインのバージョンが複数存在する場合は、`bindtextdomain()` を使用して名前空間の衝突を回避することができます。`textdomain()` を呼び出さない場合には、デフォルト時のドメインが選択されます。最後に行なった `textdomain()` の有効な呼び出しによるドメインの設定は、このあとに行う `setlocale(3C)` および `gettext()` の呼び出しにおいても有効です。

*domainname* 引数は、現在アクティブな、LC\_MESSAGES ロケールに適用されます。

*domainname* に NULL ポインタを設定して `textdomain()` を呼び出すことによって、ドメインの現在の状態に影響することなく、ドメインの現在の設定を照会することができます。*domainname* 引数に NULL 文字列を設定して `textdomain()` を呼び出すと、ドメインの設定は、デフォルト時のドメイン (`messages`) になります。

`bindtextdomain()` 関数は、メッセージドメインのパス部分 *domainname* と、*dirname* に入っている値を結合します。*domainname* が空文字列ではなく、まだ結合されていない場合には、`bindtextdomain()` は、*domainname* と *dirname* を結合します。

*domainname* が空文字列ではなく、すでに結合されている場合には、`bindtextdomain()` は、事前に行われていた結合部分を *dirname* に置き換えます。*dirname* 引数に使用できるのは、`gettext()`、`dgettext()`、または `dcgettext()` が呼び出されたときに解決される、絶対パス名または相対パス名です。*domainname* が NULL ポインタまたは空文字列の場合は、`bindtextdomain()` は NULL を返します。ユーザー定義のドメイン名は、`sys_` で始めることはできません。この文字列で始まるドメイン名は、システムで使用するために予約されています。

`bind_textdomain_codeset()` 関数を使用すると、ドメイン *domainname* のメッセージカタログ用の出力コードセットを指定できます。*codeset* 引数は、`iconv_open(3C)` 関数に対して使用可能な有効なコードセット名であるか、空ポインタである必要があります。*codeset* 引数が空ポインタの場合、`bind_textdomain_codeset()` はドメイン *domainname* 用に現在選択されているコードセットを返します。コードセットが選択されていない場合、`bind_textdomain_codeset()` は空ポインタを返します。`bind_textdomain_codeset()` 関数は複数回使用できます。同じ *domainname* 引数を指定して `bind_textdomain_codeset()` 関数を複数回使用した場合、先行する呼び出しで行われた設定は後続の呼び出しによって無効にされます。`bind_textdomain_codeset()` 関数は、選択されたコードセット名を含む文字列へのポインタを返します。この文字列は `bind_textdomain_codeset()` 関数に内部的に割り当てられているため、ユーザーが変更してはなりません。

戻り値

`gettext()`、`dgettext()`、および `dcgettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*msgid* 文字列を返します。

`ngettext()`、`dngettext()`、および `dcngettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*n* == 1 であれば、*msgid1* を返します。そうでなければ、*msgid2* を返します。

`gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` が返す文字列の個々のバイトには、空文字以外のあらゆる値が入ります。*msgid* が NULL ポインタである場合の戻り値は未定義です。返される文字列は、プログラムで修正してはならず、この後で `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` または `setlocale(3C)` を呼び出すことにより、無効になることがあります。`dgettext()`、`dcgettext()`、`dngettext()`、または `dcngettext()` に対する *domainname* 引数が NULL ポインタである場合、現在 `textdomain()` によってバインドされているドメインが使用されます。

## ngettext(3C)

`textdomain()` の正常な戻り値は、ドメインの現在の設定が入っている文字列へのポインタです。`domainname` が NULL ポインタの場合は、`textdomain()` は現在のドメインが入っている文字列へのポインタを返します。`textdomain()` を先に呼び出していない場合に `domainname` を空文字列にすると、デフォルト時のドメイン名が返されます。デフォルト時のドメイン名は、`messages` です。`textdomain()` が失敗した場合、NULL ポインタが返されます。

`bindtextdomain()` からの戻り値は、`dirname` が入った NULL で終わる文字列です。`dirname` が NULL の場合は、`domainname` と結び付けられたディレクトリバインディングです。何も結合されていない場合には、デフォルトの戻り値は `/usr/lib/locale` になります。`domainname` が NULL ポインタまたは空文字列の場合、`bindtextdomain()` は何も行わず、NULL ポインタを返します。返された文字列を呼び出し側で変更してはなりません。`bindtextdomain()` が失敗した場合、NULL ポインタが返されます。

使用法 これらのルーチンでは、メッセージの長さに制限はありません。ただし、テキスト `domainname` は、TEXTDOMAINMAX (256) バイト以内に制限されています。

ロケール変更のために `setlocale(3C)` が呼び出されていない限り、マルチスレッドアプリケーションにおいて `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()`、`textdomain()`、および `bindtextdomain()` 関数を安全に使うことができます。

`gettext()`、`dgettext()`、`dcgettext()`、`textdomain()`、および `bindtextdomain()` 関数は Solaris メッセージカタログと GNU 互換メッセージカタログの両方を処理できます。`ngettext()`、`dngettext()`、`dcngettext()`、および `bind_textdomain_codeset()` 関数は GNU 互換メッセージカタログだけを処理できます。Solaris メッセージカタログと GNU 互換メッセージカタログの詳細については、`msgfmt(1)` のマニュアルページを参照してください。

## ファイル

`/usr/lib/locale`  
メッセージ・ドメイン・ファイルのデフォルトパス部分

`/usr/lib/locale/locale/LC_MESSAGES/domainname.mo`  
言語 `locale` および `domainname` 用メッセージの入っているファイルのシステムデフォルト位置

`/usr/lib/locale/locale/LC_XXX/domainname.mo`  
`LC_XXX` が `LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` である場合に、`dcgettext()` 呼び出し用の言語 `locale` および `domainname` メッセージの入っているファイルのシステムデフォルト位置

`dirname/locale/LC_MESSAGES/domainname.mo`  
`bindtextdomain()` の正常な呼び出し後の、ドメイン `domainname` およびパス部分 `dirname` 用メッセージが入っているファイルの位置

`dirname/locale/LC_XXX/domainname.mo`  
`bindtextdomain()` の正常な呼び出し後の `LC_XXX` が、`LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` のいずれかである場合 `dcgettext()` 呼び出し用のドメイン `domainname`、言語

## ngettext(3C)

*locale*、およびパス部分 *dirname* メッセージが入っているファイルの位置

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 `msgfmt(1)`, `xgettext(1)`, `iconv_open(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

## printf(3C)

名前	printf, fprintf, sprintf, snprintf – 書式付き出力
形式	<pre>#include &lt;stdio.h&gt;  int printf(const char *format, /* args*/ ...); int fprintf(FILE *stream, const char *format, /* args*/ ...); int sprintf(char *s, const char *format, /* args*/ ...); int snprintf(char *s, size_t n, const char *format, /* args*/ ...);</pre>
機能説明	<p>printf() 関数は、標準出力ストリーム stdout 上に出力します。</p> <p>fprintf() 関数は、出力ストリーム <i>stream</i> 上に出力します。</p> <p>sprintf() 関数は、<i>s</i> から始まる連続したバイトを出力し、最後に NULL バイト (\0) をつけます。ユーザーは、十分な記憶領域が利用できることを確認する必要があります。</p> <p>snprintf() 関数は、sprintf() に引数 <i>n</i> を追加したのと同様です。<i>n</i> には <i>s</i> によって参照されるバッファサイズが指定されます。バッファは、常に NULL バイトで終了します。</p> <p>これらの各関数は、<i>format</i> の制御下で各引数の変換、書式化および出力を行います。<i>format</i> とは、初期シフトの状態 (もしあれば) で始まるか、または終わる文字列のことです。<i>format</i> は、以下に示す 0 個以上の指示語で構成されます。</p> <ul style="list-style-type: none"><li>■ 通常文字。出力ストリームに単純にコピーされる</li><li>■ 変換指定。各指定の結果として 0 個以上の引数を取り出す</li></ul> <p><i>format</i> 引数が不十分である場合、結果は未定義です。引数が残っていて、<i>format</i> が使い果たされた場合、余分な引数は評価されますが、評価されたとしても無効になります。</p> <p>変換は、引数リストの <i>format</i> のあとに続く、次の未使用の引数にではなく、<i>n</i> 番目の引数に適用されます。この場合、変換文字 % (後述を参照) は %<i>n</i>\$ シーケンスに置換されます。ここで <i>n</i> は [1, NL_ARGMAX] の範囲の 10 進数の整数で、引数リストにある引数の位置を示します。この機能によって、特定の言語に適切な順序で引数を選択する、書式文字列を定義できます (「使用例」参照)。</p> <p>書式文字列に、変換指定の書式 %<i>n</i>\$ が含まれる場合、引数リスト中の番号付けされた引数は、要求された回数だけ書式文字列から参照されます。</p> <p>書式文字列に、変換指定の書式 % が含まれる場合、引数リスト中の各引数は 1 度だけ使用されます。</p> <p>printf() 関数のすべての形式において、言語依存の小数点文字を出力文字列に挿入できます。小数点文字は、プログラムのロケール (LC_NUMERIC カテゴリ) によって定義されます。POSIX ロケールおよび小数点文字が定義されていないロケールでは、小数点文字のデフォルトはピリオド (.) です。</p>



- 変換指定 各変換指定は % 文字、または %n\$ 文字シーケンスによって導入されます。これらの文字の後には、以下に示す (順番で) 文字列が続きます。
- 変換すべき次の引数を指定する、最後に \$ が付いた 10 進数の文字列のフィールド (任意)。このフィールドがない場合は、最後に変換された引数の次の *args* が変換されます。
  - 変換指定の意味を変更する 0 個以上の *flags* (順不同)。
  - *field width* の最小値 (任意)。変換された値がフィールド幅より少ないバイトである場合は、デフォルトとしてフィールド幅の左側に (以下で説明する左位置合わせフラグ (-) が指定されている場合は右側に)、空白をパディング (文字詰め) します。フィールド幅は、アスタリスク (\*) (以下に説明) または 10 進数の整数の形式を取ります。  
変換文字が s の場合、標準準拠のアプリケーション (standards(5) を参照) は、フィールド幅を、出力される場合の最小バイト数として解釈します。また、標準準拠ではないアプリケーションは、フィールド幅を、画面表示の最小カラム数として解釈します。標準準拠ではないアプリケーションに対しては、たとえば %10s とは、変換された値が 7 カラムの画面幅の場合に右側に 3 個分の空白がパディングされることを意味します。  
書式が %ws ならば、フィールド幅を画面表示の最小カラム数として解釈します。
  - 精度を指定する *precision* (任意)。d、i、o、u、x、および X 変換の場合、数値の最小桁数を表します。この場合、このフィールドの先頭は 0 でパディングされます。e、E、および f 変換の場合、小数点文字以下の数字の桁数を表します。g および G 変換の場合、最大有効桁数を表します。また、s および S 変換の文字列からは、最大バイト数が出力されます。精度は、最初にピリオド (.) が来て、そのあとにアスタリスク (\*) (以下に説明) または 10 進数の文字列 (任意) が続く形式になります。NULL の 10 進数の文字列は 0 として扱われます。精度がその他の変換文字で表される場合、動作は未定義です。  
変換文字が s または S の場合、標準準拠のアプリケーション (standards(5) を参照) は、精度を、出力される場合の最大バイト数として解釈します。また、標準準拠ではないアプリケーションは、精度を、画面表示の最大カラム数として解釈します。標準準拠ではないアプリケーションに対しては、たとえば %.5s は、画面の 5 カラムに表示する文字列部分だけを出力します。完全な文字だけが書き込まれません。  
%ws の場合、精度を画面表示の最大カラム数として解釈します。精度は、最初にピリオドが来て、その後 10 進数の文字列が続く形式になります。NULL の 10 進数の文字列は 0 として扱われます。精度で指定したパディングは、フィールド幅で指定したパディングより優先されます。
  - h (任意)。変換文字 d、i、o、u、x、および X が short int 型または unsigned short int 型の引数に適用されることを指定します (引数は、整数変換 (インテグラルプロモーション) の規則に従って変換されます。値は short int または unsigned short int の形式に変換されてから表示されます)。また h は、変換文字 n が short int 型の引数へのポインタに適用されることを指定します。l (小文字のエル) (任意指定) は、変換文字 d、i、o、u、x、および X が long int 型または unsigned long int 型の引数に適用されることを指定します。また l (小文字のエル) は、変換文字 n が long int 型の引数へのポインタに適用されることを指定します。ll (小文字のエルが 2 つ) (任意指定) は、変換文字

## printf(3C)

d、i、o、u、x、および X が long long 型または unsigned long long 型の引数に適用されることを指定します。また ll (小文字のエルが 2 つ) は、変換文字 n が long long 型の引数へのポインタに適用されることを指定します。L (任意) は、変換文字 e、E、f、g、および G が long double 型の引数に適用されることを指定します。h、l、および L が上記以外の変換文字に現われたときの変換結果は、未定義です。

- l (小文字のエル) (任意)。変換文字 c が wint\_t 引数に適用されることを指定します。また L (大文字のエル) は、変換文字 s が wint\_t 引数へのポインタに適用されることを指定します。
- 適用される変換の型を意味する *conversion character* (下記参照)。

フィールド幅および精度の両方またはどちらか一方は、アスタリスク (\*) で示すこともできます。この場合、int 型の引数によってフィールド幅または精度を指定します。フィールド幅および精度の両方またはどちらか一方を指定する引数 (もしあれば) は、変換される順番で引数の前に現れなければなりません。フィールド幅が負の値の場合、- フラグの後に正のフィールド幅を指定したように扱われます。精度が負の値の場合、精度を省略したように扱われます。%n\$ という形式で変換を指定している書式文字列では、フィールド幅または精度は \*m\$ シーケンスで示すことができます。この場合 m は、[1, NL\_ARGMAX] の範囲の 10 進数の整数で、整数引数の引数リスト内 (書式引数に続く) のフィールド幅または精度の位置を示します。以下に例を示します。

```
printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

*format* には、番号付けされた引数指定 (つまり %n\$ と \*m\$)、または番号付けされていない引数指定 (つまり % と \*) のいずれかを含むことができます。通常、両方を指定することはできません。唯一の例外として、%% と %n\$ の形式の両方を含むことがあります。*format* 文字列に、番号付けされた引数と番号付けされていない引数とを、同時に指定した場合の結果は未定義です。番号付けされた引数を指定するとき、N 番目の引数を指定する場合は、最初から (N-1) 番目までの先行引数を書式文字列内で指定する必要があります。

フラグ文字 以下に、フラグ文字とその意味を説明します。

- |           |   |
|-----------|---|
| '         | 10 進数変換 (%i、%d、%u、%f、%g、または %G) の結果の整数部分が、千単位のグループ化文字でフォーマットされます。その他の変換についての動作結果は未定義です。貨幣以外のグループ化文字が使用されます。     |
| -         | 変換の結果は、フィールド内で左詰めされます。このフラグを指定しない場合、変換の結果は右詰めされます。  |
| +         | 符号つき変換の結果は、常に符号 (+ または -) で始まります。このフラグを指定しない場合、負の値を変換するときだけ符号で始まります。  |
| 空白 (スペース) | 符号つき変換の最初の文字が符号でない場合または変換の結果に文字がない場合は、結果の前に空白がおかれます。つまり、空白のフラグと + フラグを共に指定した場合、空白のフラグは無視されます。                   |
| #         | 値は代替形式に変換されます。c、d、i、s、および u 変換の場合、このフラグは影響しません。o 変換の場合、このフラグは (必要な場合には) 精度をあげて、結果の最初の数字をゼロにします。x (または X) 変換では、0 |

以外の結果の前に 0x (または 0X) が追加されます。e、E、f、g、および G 変換の場合、必ず、結果は小数点付き数字になります。小数点の後に数字が続かない場合でも小数点が付きます (このフラグが指定されていない場合、これらの変換の結果では、小数点以降の数字がある場合のみ小数点がつけられます)。g および G 変換の場合、結果から後続 0 が削除されません (通常は削除されます)。

0 d, i, o, u, x, X, e, E, f, g, および G 変換の場合、先行 0 を (符号または基数の後ろに) 用いて、フィールド幅をパディングします。空白がパディングされることはありません。0 フラグおよび - フラグをともに指定すると、0 フラグが無視されます。d、i、o、u、x、および X の各変換では、精度を指定すると、0 フラグは無視されます。0 フラグおよび ' フラグをともに指定すると、0 パディングの前に文字が挿入されます。他の変換では、このフラグの動作は未定義です。

変換文字 各変換文字を指定すると、0 個以上の引数を取り出されます。書式に関する引数の指定が不十分な場合の変換結果は、未定義です。書式が終了しても引数が残っている場合は、残りの引数は無視されます。

変換文字とその意味を以下で説明します。

d,i int 引数は、[-]dddd の形式で符号付き 10 進数に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。

o unsigned int 引数は、dddd の形式で符号なし 8 進数の書式に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。

u unsigned int 引数は、dddd の形式で符号なし 10 進数の書式に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。

x unsigned int 引数は、dddd の形式で符号なし 16 進数の書式に変換されます。abcdef 文字が使用されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表せる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。

X ABCDEF 文字が abcdef 文字の代わりに使用される点を除いて、x 変換文字と同じ動作です。

f double の引数は、[-]ddd.ddd の形式で 10 進数に変換されます。小数点付き数字 setlocale(3C) 参照) の後の桁数は、精度で指定した数です。精度が指定されていない場合は、6 とみなされます。精度を明示的に 0 に指定

## printf(3C)

	<p>し、# フラグを指定しない場合、小数点付き数字は出力されません。小数点付き数字が出力されると、1 個以上の数字がその前に付きます。値は適切な桁数に丸められます。</p>
e,E	<p>double の引数は、[-]d.ddde±dd の形式に変換されます。小数点付き数字の前には 1 個の数字が付きます (引数が 0 でない場合はこの数字も 0 ではありません)。小数点付き数字の後には、精度で指定した数の数字が続きます。精度を指定しない場合は、6 とみなされます。精度が 0 で、# フラグを指定しない場合、小数点付き数字は出力されません。E 変換文字を使用すると、e ではなく E を数字につけて指数を示します。指数は、必ず 2 桁以上の数字です。値は適切な桁数に丸められます。</p>
g,G	<p>double の引数は、f または e の形式 (G 変換文字の場合は E 形式) で出力されます。精度は有効桁数を示します。明示的な精度が 0 である場合は、有効桁数が 1 桁になります。出力形式は変換される値によって決まります。変換の結果出力される指数が -4 より小さい場合、または精度以上の場合だけ、e (または E) 形式で出力されます。結果の小数部分からは後続 0 が取り除かれます。小数点付き数字は、その後に数字が続く場合だけ出力されます。</p>
c	<p>int の引数は、unsigned char 符号なし文字に変換され、結果のバイトが出力されます。</p> <p>l (小文字のエル) を修飾子として指定した場合、wint_t 引数は、精度は付けなくて、wchar_t 型の 2 つの要素の配列をポイントする引数を付けた ls 変換指定を行なった場合と同じように変換されます。最初の要素には ls 変換指定への wint_t 引数を含み、2 番目の要素には NULL ワイド文字を含みます。</p>
C	<p>lc と同じです。</p>
wc	<p>int の引数はワイド文字 (wchar_t) に変換されて出力されます。</p>
s	<p>引数は char 配列へのポインタにする必要があります。配列の中身が最後の NULL バイトまで書き込まれます (NULL バイトは含まない)。精度を指定すると、標準準拠のアプリケーション (standards(5) を参照) は、精度によって指定されるバイト数だけを書き込みます。また、標準準拠ではないアプリケーションは、精度によって指定された表示画面の カラム数に表示される文字列部分だけを出力します。精度を指定しないと場合は、無限とみなされ、最初の NULL バイトまでのすべてのバイトが出力されます。値が NULL の引数の変換は、未定義の結果になります。</p> <p>修飾子として l (小文字のエル) を指定した場合、引数は wchar_t 型の並びへのポインタである必要があります。その並びからのワイド文字コードは、NULL ワイド文字コードに至るまで、末尾に NULL ワイド文字コードを含まない文字に変換されます (最初のワイド文字が変換される前に、mbstate_t オブジェクトを 0 に初期化して記述された変換状態の、wrtomb(3C) 関数への呼び出しによる結果と同じように変換されます)。結果として生じる文字は、末尾の NULL 文字 (バイト) に至るまで (ただし、末尾には含まない) で書き込まれます。精度を指定しない場合、</p>

配列には、NULL ワイド文字を含む必要があります。精度を指定した場合、指定した文字 (バイト) 以上は書き込まれません (もしあれば、シフトシーケンスを含む)。配列が NULL ワイド文字を含む必要がある場合は、精度によって指定された文字シーケンスの長さを書き込み、関数は、一度配列の最後を過ぎたワイド文字へはアクセスを必要としません。文字の一部を書き込むことはありません。

s ls と同じです。

ws 引数は `wchar_t` 配列へのポインタの文字列にする必要があります。その配列の中身が最後の NULL 文字まで書き込まれます (NULL 文字は含まない)。精度を指定すると、精度によって指定された表示画面の列数に表示するワイド文字列部分だけを出力します。精度を指定しない場合は無限とみなされ、最初の NULL 文字までのすべてのワイド文字が出力されます。値が NULL の引数を指定した場合の変換は、未定義の結果になります。

p 引数は `void` へのポインタにする必要があります。ポインタの値は、出力可能なシーケンスに変換されます。これらの文字は、`scanf(3C)` 関数の `%p` 変換で一致した文字と同じ文字である必要があります。

n 引数は、`printf()` 関数のうちの 1 つの呼び出しにおいて、この変換文字が指定されるまでに、出力先に書き込まれたバイト数が格納される整数へのポインタです。引数は変換されません。

% % を出力します。引数は変換されません。全体の変換を指定するには `%%` にしてください。

変換指定が上記の形式のいずれにも当てはまらない場合、変換の結果は未定義となります。

浮動小数点値によって「無限」を内部的に表現している場合は、`[±]Infinity` が出力されます。希望した出力文字列の長さに応じて、`Infinity` は `Infinity` または `Inf` のどちらかになります。符号は、前述の規則に従って出力されます。

浮動小数点値によって「数でない値」を内部的に表現している場合は、`[±]NaN` が出力されます。符号は、前述の規則に従って出力されます。

フィールド幅が存在していなかったり、あるいはその値が小さい場合でも、フィールドが切り捨てられることはありません。変換の結果がフィールド幅よりも広い場合は、その結果を収容できるようにフィールドが拡張されます。`printf()` および `fprintf()` によって生成される文字は、`putc(3C)` 関数を呼び出した場合と同じように出力されます。

ファイルの `st_ctime` と `st_mtime` フィールドは、`printf()` または `fprintf()` の正常実行への呼び出しと、同じストリーム上の `fflush(3C)` または `fclose(3C)` あるいは `exit(3C)` または `abort(3C)` への呼び出しへの次の正常終了の呼び出しとの間で、更新するためにマークされます。

戻り値 `printf()`、`fprintf()`、および `sprintf()` 関数は、転送されるバイト数 (`sprintf()` の場合は最後の NULL バイトを除く) を返します。

## printf(3C)

	<p>snprintf() 関数はフォーマットされた文字数、つまり (バッファに十分な大きさがある場合に) バッファに書き込まれた文字数を返します。snprintf() への呼び出し時に <i>n</i> の値が 0 である場合、1 未満の指定されていない値が返されます。</p> <p>出力エラーが検出された場合、各関数は負の値を返します。</p>
エラー	<p>printf() と fprintf() の両関数が異常終了する、または異常終了する可能性がある条件については、fputc(3C) または fputwc(3C) を参照してください。</p> <p>さらに printf() のすべての形式において、以下の条件で異常終了します。</p> <p>EILSEQ           有効な文字に対応しないワイド文字コードが検出された。</p> <p>EINVAL            引数が足りない</p> <p>さらに printf() と fprintf() は、以下の条件で異常終了します。</p> <p>ENOMEM            使用可能な記憶領域が不足している</p>
使用法	<p>printf() 関数の呼び出しに wint_t または wchar_t 型のオブジェクトが存在する場合、これらのオブジェクトを定義するヘッダー &lt;wchar.h&gt; も含む必要があります。</p> <p>sprintf() と snprintf() の両関数はマルチスレッドアプリケーションでは MT-Safe です。ロケール変更のために setlocale(3C) が呼び出されていない限り、マルチスレッドアプリケーションにおいて printf() および fprintf() 関数を安全に使用できます。</p>
エスケープシーケンス	<p>通常 printf() 関数に対する書式化文字列を入力するときには、C 言語に組み込まれている以下のエスケープシーケンスを使用します。ただし、これらのエスケープシーケンスは、printf() 関数によってではなく C コンパイラによって処理されます。</p> <p>\a            ベルを鳴らして警報を出します。</p> <p>\b            バックスペース。現在位置が行頭でなければ、出力位置を現在位置の 1 文字前に移動させます。</p> <p>\f            フォームフィード。出力位置を次の論理ページの最初の出力位置に移動させます。</p> <p>\n            改行。出力位置を次の行の行頭に移動させます。</p> <p>\r            キャリッジリターン。出力位置を現在行の行頭に移動させます。</p> <p>\t            水平タブ。出力位置を、現在行上の次の水平ハードタブ位置に移動させます。</p> <p>\v            垂直タブ。出力位置を、垂直ハードタブ位置の始めに移動させます。</p> <p>また C 言語では、次の形式の文字シーケンスがサポートされています。</p> <p>\octal-number および</p>

## printf(3C)

`\hex-number` 8進数 `octal-number` または 16進数 `hex-number` で表す文字に変換します。たとえば、ASCII 表現の場合、文字 `a` は `\141` として書き込まれ、文字 `Z` は `\132` として書き込まれます。この文法は、NULL 文字 `\0` を表す場合に最もよく使用されます。これは定数ゼロ (0) とまったく同等です。通常の 8進数と同様に、8進数に接頭辞ゼロが含まれていないことに注意してください。16進数を指定するには、接頭辞の `0` を取り除いて接頭辞が `x` (小文字) になるようにします (大文字の `X` は不可です)。16進シーケンスのサポートは、ANSI の拡張機能です。standards(5) を参照してください。

**使用例** 例 1 言語に依存する日付と時刻の形式を出力するには、以下のステートメントを使用します。

```
printf (format, weekday, month, day, hour, min);
```

米国の使用法では、*format* は次の文字列へのポインタになります。

```
"%s, %s %d, %d:%.2d\n"
```

生成されるメッセージは次のとおりです。

```
Sunday, July 3, 10:02
```

ドイツの使用法では、*format* は次の文字列へのポインタになります。

```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

生成されるメッセージは次のとおりです。

```
Sonntag, 3. Juli, 10:02
```

例 2 `Sunday, July 3, 10:02` の形式で日付と時刻を出力し、`weekday` と `month` が NULL で終わる文字列へのポインタにする場合は以下ようになります。

```
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);
```

例 3 小数点以下 5 桁まで出力する場合は以下ようになります。

```
printf("pi = %.5f", 4 * atan(1.0));
```

**デフォルト** 例 4 標準準拠ではないアプリケーション (standards(5) を参照) での `printf()` の動作だけに適用します。20 文字幅で名前のリストを表示する場合は以下ようになります。

```
printf("%20s%20s%20s", lastname, firstname, middlename);
```

**属性** 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

printf(3C)

関連項目 exit(2), lseek(2), write(2), abort(3C), ecvt(3C), exit(3C), fclose(3C),  
fflush(3C), fputwc(3C), putc(3C), scanf(3C), setlocale(3C), stdio(3C),  
wcstombs(3C), wctomb(3C), attributes(5), environ(5), standards(5)



名前	fputc, putwc, putwchar – ワイド文字コードをストリームへ書き出す
形式	<pre>#include &lt;stdio.h&gt; #include &lt;wchar.h&gt;  wint_t fputc(wchar_t wc, FILE*stream); wint_t putwc(wchar_t wc, FILE*stream);  #include &lt;wchar.h&gt; wint_t putwchar(wchar_t wc);</pre>
機能説明	<p><b>fputc()</b> fputc() 関数は、ワイド文字コード <i>wc</i> に対応する文字を、<i>stream</i> が指す出力ストリーム中に書き出します。書き出す位置は、出力ストリーム用のファイルポジション表示 (定義されていれば) が指す位置です。出力後、ファイルポジション表示の値を進めます。ファイルが位置指定要求をサポートしていない場合、またはストリームが追加モードでオープンされていた場合には、文字は出力ストリームに追加されます。書き込み中にエラーが発生すると、出力ファイルのシフト状態は未定義のままになります。</p> <p>ファイル中の <i>st_ctime</i> と <i>st_mtime</i> の両フィールドは、fputc() が正常終了してから、同じストリームに対する次の fflush(3C) または fclose(3C) の呼び出しが正常終了するまで、もしくは exit(2) または abort(3C) が呼び出されるまでの間に更新されるようマークされます。</p> <p><b>putwc()</b> putwc() 関数は、マクロとして実装される点を除いては、fputc() 関数と同じです。</p> <p><b>putwchar()</b> putwchar(<i>wc</i>) の呼び出しは、putwc(<i>wc</i>, stdout) と同じです。putwchar() ルーチンはマクロとして実装されます。</p> <p>戻り値 正常終了時、fputc()、putwc()、および putwchar() は <i>wc</i> を返します。正常終了しない場合は WEOF を返します。このとき、ストリームのエラー表示がセットされ、errno はエラーを示す値に設定されます。</p> <p>エラー ストリームが蓄積されていないか、あるいはストリームのバッファ中のデータを書き出す必要があるとき、fputc()、putwc()、および putwchar() 関数は以下の条件で異常終了します。</p> <p>EAGAIN <i>stream</i> に対応するファイル記述子に O_NONBLOCK フラグが設定されていて、プロセスの動作が書き込みで待たされる可能性がある</p> <p>EBADF <i>stream</i> に対応するファイル記述子が、書き込み用にオープンされた正しいものでない。または、ファイルは通常ファイルで、対応する <i>stream</i> に関連するオフセット最大値、またはそれ以上で作成しようとした</p> <p>EINTR シグナルを受け取り、書き込みが終了した。データは転送されていない</p>

## putc(3C)

EIO	物理 I/O エラーが発生した。または、プロセスがバックグラウンドプロセスのグループで、そのグループが制御端末に書き込もうとしていて、TOSTOP が設定されており、プロセスが SIGTTOU を無視もブロックもせず、さらにプロセスグループの親がなくなっている				
ENOSPC	ファイルが存在するデバイス上に、空き領域がない				
EPIPE	プロセスによる読み込み用にオープンされていないパイプまたは FIFO へ書き込もうとした。この場合、SIGPIPE シグナルもプロセスへ送られる				
fputc(), putc(), および putwchar() 関数は以下の条件で異常終了します。					
ENOMEM	使用可能な記憶領域が十分でない				
ENXIO	存在しないデバイスに対して要求が出された、またはデバイスの能力を超えた要求が出された				
EILSEQ	ワイド文字コード <i>wc</i> が、正当な文字に対応していない				
使用法	putc() および putwchar() のマクロには、関数が存在します。関数のフォームを得るには、マクロ名を未定義にする必要があります (たとえば #undef putc)。  マクロのフォームを使用した場合、putc() は <i>stream</i> 引数を 2 回以上、評価します。特に putc( <i>wc</i> , *f++) は正しく動作しません。 <i>stream</i> 引数に影響するような評価をする場合は、代わりに fputc() 関数を使用するようにしてください。				
属性	次の属性については attributes(5) のマニュアルページを参照してください。				
<table border="1"> <thead> <tr> <th>属性タイプ</th> <th>属性値</th> </tr> </thead> <tbody> <tr> <td>MT レベル</td> <td>MT-Safe</td> </tr> </tbody> </table>		属性タイプ	属性値	MT レベル	MT-Safe
属性タイプ	属性値				
MT レベル	MT-Safe				
関連項目	exit(2), ulimit(2), abort(3C), fclose(3C), ferror(3C), fflush(3C), fopen(3C), setbuf(3C), attributes(5)				

名前	fputc, putwc, putwchar – ワイド文字コードをストリームへ書き出す
形式	<pre>#include &lt;stdio.h&gt; #include &lt;wchar.h&gt;  wint_t fputc(wchar_t wc, FILE*stream); wint_t putwc(wchar_t wc, FILE*stream);  #include &lt;wchar.h&gt; wint_t putwchar(wchar_t wc);</pre>
機能説明	<p><b>fputc()</b> fputc() 関数は、ワイド文字コード <i>wc</i> に対応する文字を、<i>stream</i> が指す出力ストリーム中に書き出します。書き出す位置は、出力ストリーム用のファイルポジション表示 (定義されていれば) が指す位置です。出力後、ファイルポジション表示の値を進めます。ファイルが位置指定要求をサポートしていない場合、またはストリームが追加モードでオープンされていた場合には、文字は出力ストリームに追加されます。書き込み中にエラーが発生すると、出力ファイルのシフト状態は未定義のままになります。</p> <p>ファイル中の <i>st_ctime</i> と <i>st_mtime</i> の両フィールドは、fputc() が正常終了してから、同じストリームに対する次の fflush(3C) または fclose(3C) の呼び出しが正常終了するまで、もしくは exit(2) または abort(3C) が呼び出されるまでの間に更新されるようマークされます。</p> <p><b>putwc()</b> putwc() 関数は、マクロとして実装される点を除いては、fputc() 関数と同じです。</p> <p><b>putwchar()</b> putwchar(<i>wc</i>) の呼び出しは、putwc(<i>wc</i>, stdout) と同じです。putwchar() ルーチンはマクロとして実装されます。</p> <p>戻り値 正常終了時、fputc()、putwc()、および putwchar() は <i>wc</i> を返します。正常終了しない場合は WEOF を返します。このとき、ストリームのエラー表示がセットされ、errno はエラーを示す値に設定されます。</p> <p>エラー ストリームが蓄積されていないか、あるいはストリームのバッファ中のデータを書き出す必要があるとき、fputc()、putwc()、および putwchar() 関数は以下の条件で異常終了します。</p> <p>EAGAIN <i>stream</i> に対応するファイル記述子に O_NONBLOCK フラグが設定されていて、プロセスの動作が書き込みで待たされる可能性がある</p> <p>EBADF <i>stream</i> に対応するファイル記述子が、書き込み用にオープンされた正しいものでない。または、ファイルは通常ファイルで、対応する <i>stream</i> に関連するオフセット最大値、またはそれ以上で作成しようとした</p> <p>EINTR シグナルを受け取り、書き込みが終了した。データは転送されていない</p>

## putwchar(3C)

EIO	物理 I/O エラーが発生した。または、プロセスがバックグラウンドプロセスのグループで、そのグループが制御端末に書き込もうとしていて、TOSTOP が設定されており、プロセスが SIGTTOU を無視もブロックもせず、さらにプロセスグループの親がなくなっている				
ENOSPC	ファイルが存在するデバイス上に、空き領域がない				
EPIPE	プロセスによる読み込み用にオープンされていないパイプまたは FIFO へ書き込もうとした。この場合、SIGPIPE シグナルもプロセスへ送られる				
fputc(), putwc(), および putwchar() 関数は以下の条件で異常終了します。					
ENOMEM	使用可能な記憶領域が十分でない				
ENXIO	存在しないデバイスに対して要求が出された、またはデバイスの能力を超えた要求が出された				
EILSEQ	ワイド文字コード <i>wc</i> が、正当な文字に対応していない				
使用法	putwc() および putwchar() のマクロには、関数が存在します。関数のフォームを得るには、マクロ名を未定義にする必要があります (たとえば #undef putc)。  マクロのフォームを使用した場合、putwc() は <i>stream</i> 引数を 2 回以上、評価します。特に putwc( <i>wc</i> , *f++) は正しく動作しません。 <i>stream</i> 引数に影響するような評価をする場合は、代わりに fputc() 関数を使用するようにしてください。				
属性	次の属性については attributes(5) のマニュアルページを参照してください。				
<table border="1"> <thead> <tr> <th>属性タイプ</th> <th>属性値</th> </tr> </thead> <tbody> <tr> <td>MT レベル</td> <td>MT-Safe</td> </tr> </tbody> </table>		属性タイプ	属性値	MT レベル	MT-Safe
属性タイプ	属性値				
MT レベル	MT-Safe				
関連項目	exit(2), ulimit(2), abort(3C), fclose(3C), ferror(3C), fflush(3C), fopen(3C), setbuf(3C), attributes(5)				

名前 putws – ワイド文字列の EUC 文字への変換

形式 

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

int putws(wchar_t *s);
```

機能説明 putws() 関数は、s が指す ((wchar\_t) NULL で終わる) ワイド文字の文字列を、拡張 UNIX コード (EUC) 文字列に変換し、復帰改行を加え、標準出力ストリーム stdout に書き込みます。文字列の最後の NULL 文字は書き込みません。

戻り値 putws() 関数は、変換され書き込まれたワイド文字の文字数を返します。書き込み用にオープンされていないファイルに書き込もうとした場合、EOF を返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 ferror(3C), fopen(3C), fread(3C), getws(3C), printf(3C), putwc(3C), attributes(5)

## scanf(3C)

名前	scanf, fscanf, sscanf, vscanf, vfscanf, vsscanf – 書式付き入力の変換
形式	<pre> #include &lt;stdio.h&gt;  int scanf(const char *format, ...); int fscanf(FILE*stream, const char *format, ...); int sscanf(const char *s, const char *format, ...);  #include &lt;stdarg.h&gt; #include &lt;stdio.h&gt;  int vscanf(const char *format, va_list arg); int vfscanf(FILE *stream, const char *format, va_list arg); int vsscanf(const char *s, const char *format, va_list arg); </pre>
機能説明	<p>scanf() 関数は、標準入力ストリーム stdin を読み取ります。</p> <p>fscanf() 関数は、入力ストリーム <i>stream</i> を読み取ります。</p> <p>sscanf() 関数は、文字列 <i>s</i> を読み取ります。</p> <p>vscanf(), vfscanf(), および vsscanf() 関数はそれぞれ、scanf(), fscanf(), および sscanf() と同等ですが、これらの関数を呼び出すときは、可変数個の引数ではなく、&lt;stdarg.h&gt; ヘッダーで定義されている引数リスト (stdarg(3HEAD) のマニュアルページを参照) を使用します。これらの関数は va_end() マクロを呼び出しません。そのため、アプリケーションは、これらの関数を使用した後、va_end(ap) を呼び出してクリーンアップ作業を行う必要があります。</p> <p>これらの関数は、それぞれ、バイトを読み取り、書式にしたがってそれを解釈し、その結果を引数に格納します。これらの関数には、引数として、制御文字列 <i>format</i> (以下で説明)、および変換された入力の格納場所を示す <i>pointer</i> 群を指定します。書式に対して十分な引数が指定されていない場合、変換の結果は未定義となります。書式が終了しても引数が残っている場合、残りの引数は評価されますが、評価されたとしても無視されます。</p> <p>変換は、次の未使用の引数にではなく、引数リストの <i>format</i> のあとに続く <i>n</i> 番目の引数に適用されます。この場合、変換文字 % (後述を参照) は %n\$ シーケンスに置換されます。ここで <i>n</i> は [1, NL_ARGMAX] の範囲の 10 進数の整数です。この変換機能は、特定の言語に合わせた順序で引数を選択するように、書式文字列の定義を規定します。書式文字列に変換指定の書式 %n\$ が含まれる場合、引数リスト中の番号付けされた引数が、複数回、書式文字列から参照されるかどうかについては不定です。</p> <p><i>format</i> には、変換指定の形式、つまり % または %n\$ のいずれかを含むことができます。ただし、通常は 1 つの <i>format</i> 文字列に % と %n\$ の両方を指定することはできません。唯一の例外として、%% または %* を %n\$ の形式に入れることがあります。</p>

scanf() 関数のすべての形式において、入力文字列中の言語依存の小数点文字を検出できます。小数点文字は、プログラムのロケール(LC\_NUMERIC カテゴリ)によって定義されます。POSIX ロケール および小数点文字が定義されていないロケールでは、小数点文字のデフォルトはピリオド(.)です。

*format* とは、初期シフトの状態が始まるかまたは終わる文字列で、0 個以上の指示語(もしあれば)で構成されます。各指示語は次のいずれか 1 つで構成されます。

- 1 つまたは複数の空白文字(スペース、タブ、改行文字、垂直タブ、または用紙送り(form-feed)文字)
- 通常文字(% または空白文字は含まない)
- 変換指定

#### 変換指定

各変換指定は % 文字、または %n\$ 文字シーケンスによって導入されます。これらの文字の後には、以下に示す(順番で)文字列が続きます。

- 代入抑制文字 \* (任意)
- 最大フィールド幅を指定する 0 以外の 10 進数 (任意)
- 受け取るオブジェクトのサイズを表すサイズ修飾子、h、l (小文字のエル)、ll (小文字のエル 2 つ)、または L (任意)。対応する引数が int ではなく short int へのポインタである場合は、変換文字 d、i、および n の前に h を指定する必要があります。また、対応する引数が long int へのポインタである場合は、変換文字 d、i、および n の前に l (小文字のエル) を指定する必要があります。対応する引数が long long int へのポインタである場合は、変換文字 d、i、および n の前に ll (小文字のエル 2 つ) を指定する必要があります。同様に、対応する引数が unsigned int へのポインタではなく unsigned short int へのポインタである場合は、変換文字 o、u、および x の前に h を指定する必要があります。また、対応する引数が unsigned long int へのポインタである場合は、変換文字 o、u、および x の前に l (小文字のエル) を指定する必要があります。対応する引数が unsigned long long int へのポインタである場合は、変換文字 o、u、および x の前に ll (小文字のエル 2 つ) を指定する必要があります。対応する引数が float へのポインタではなく double へのポインタである場合は、変換文字 e、f、および g の前に l (小文字のエル) を指定する必要があります。または、対応する引数が long double へのポインタである場合は、変換文字 e、f、および g の前に L を指定する必要があります。最後に、対応する引数が 文字形式へのポインタではなく wchar\_t へのポインタである場合は、変換文字 c、s、および [ の前に l (小文字のエル) を指定する必要があります。h、l (小文字のエル)、ll (小文字のエル 2 つ)、および L が上記以外の変換文字の前に現われたとき、変換の結果は未定義となります。
- 適用される変換形式を指定する変換文字 (有効な変換文字については、後述の説明を参照してください)。

scanf() 関数は、形式の各指示を順番に実行します。指示にエラーがあった場合、以下に詳しく記述しているように、関数はエラーを返します。エラーは、入力エラー(入力バイトが無効)またはマッチングエラー(入力が不相当)として記述されます。

## scanf(3C)

1 つまたは複数の空白文字で構成される指示は、有効な入力を読み取られなくなるまで、あるいは空白文字でないために読み取れずに残る最初のバイトまで、入力を読み取って実行します。

通常文字の指示は次のように実行されます。次のバイトが入力から読み取られ、指示を含むバイトと比較されます。比較の結果、両者が同じでない場合、指示はエラーで終了し、違いのあった後続のバイトが読み取れずに残ります。

変換指定を表す指示は、一連のマッチング入力シーケンスを定義します (各変換文字については以下に記述)。変換指定は次の手順で実行されます。

変換指定に変換文字 `l`、`c`、`C`、または `n` が含まれる場合を除いて、空白文字の入力 (`isspace(3C)` で指定) はスキップされます。

変換指定に変換文字 `n` が含まれる場合を除いて、項目は入力から読み取られます。入力項目は、入力バイトの最も長いシーケンスとして (変換文字によっては文字またはバイト単位で測定される、指定された最大フィールド幅まで) 定義されます。入力項目幅はマッチングシーケンスの最初のサブシーケンスです。入力項目に続く最初のバイト (もしあれば) は、読み取れずに残ります。入力項目の長さが 0 の場合、変換指定の実行はエラーで終了します。この状態は、ファイルの終わり、コード化エラー、あるいはストリームからの入力を妨げる読み取りエラー (この場合は入力エラー) でない限り、マッチングのエラーです。

変換文字が `%` ではない場合の入力項目 (あるいは変換指定が `%n` の場合の入力バイト数) は、変換文字に適切な形式に変換されます。入力項目がマッチングシーケンスではない場合、変換指定はエラーで終了します。この状態はマッチングエラーです。代入抑制文字が `*` で表された場合を除いて、変換の結果は、以前に変換結果を受け取っていない `format` 引数に続く最初の引数によって、ポイントされたオブジェクトに出力されます (変換指定が `%` によって呼び出されていない場合)。変換指定が文字シーケンス `%n$` によって呼び出された場合は、`n` 番目の引数に出力されます。このオブジェクトが適切な形式ではない場合、または変換の結果が提供された空間に対応できない場合、動作結果は未定義です。

変換文字 以下に、有効な変換文字を示します。

- d 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- i 整数 (符号は任意) と一致します。書式は、`strtoul()` の `base` 引数に値 0 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- o 8 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 8 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。



- u 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- x 16 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 16 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- e,f,g 符号付き浮動小数点数に一致します。書式は `strtod(3C)` の変換対象となる文字コードの並びと同じです。サイズ修飾子を指定しない場合、対応する引数は `float` へのポインタである必要があります。
- `printf(3C)` ファミリの関数が、ANSI/IEEE Std 754: 1985 標準を指定できるように、無限および NaN (浮動小数点形式にコード化された 7858 シンボリックエンティティ) 用に文字列表現を生成した場合、`scanf()` ファミリの関数はそれらを入力として認識します。
- s 空白でないバイトシーケンスに一致します。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- 1 (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。それぞれの文字は、`mbrtowc(3C)` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- [ 複数の文字 (*scanset* という) からなる、空でない文字シーケンスと一致します。この場合、通常行われる先行する空白のスキップは抑制されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL バイトを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- 1 (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- 変換指定は、*format* 文字列のすべての後続の文字、つまり、対応する右角括弧 (]) まで (または、その括弧も含む) が入ります。括弧内の文字 (*scanlist*) は *scanset* を構成しています。左角括弧の次の文字がサーカンフ

## scanf(3C)

レックス (^) の場合は、サーカンフレックスと直角括弧の間 *scanlist* に入っていないすべての文字が *scanset* を構成しています。変換指定が [ ] または [^] で始まる場合は、この直角括弧は *scanlist* 内に含まれており、次の直角括弧が指定の終了を示す直角括弧になります。*scanlist* に - があり、その - が最初の文字ではない場合、最初の文字が ^ で - が 2 番目の文字ではない場合、あるいは - が最後の文字ではない場合、一致する文字の範囲が表示されます。

**c** フィールド幅に指定された数 (フィールド幅が変換指定にない場合は 1) の文字シーケンスと一致します。対応する引数は、その文字シーケンスを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。NULL 文字は追加されません。この場合、通常行われる先行する空白のスキップは抑制されます。

l (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよび最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。NULL ワイド文字は追加されません。

**p** 対応する `printf(3C)` 関数の `%p` 変換によって出力されるシーケンスと同様のシーケンスと一致します。対応する引数は `void` へのポインタである必要があります。入力項目が、同一プログラムの実行中で以前に変換した値である場合は、ポインタは先にその値を変換したときと同様の比較を行います。そうでない場合は、`%p` 変換の動作は未定義です。

**n** 入力は使用 (変換) されません。対応する引数は、これまでに `scanf()` 関数を呼び出して入力ストリームから読み取ったバイト数を示す整数へのポインタである必要があります。変換指定 `%n` を実行しても、この関数の実行終了時に返される代入数は増加しません。

**C** `lc` と同じです。

**S** `ls` と同じです。

**%** 単一の `%` と一致します。変換または代入は行われません。完全な変換指定をするには `%%` を指定してください。

変換指定が正しくない場合の動作は未定義です。

変換文字 `E`, `G`, および `X` はそれぞれ `e`, `g`, および `x` と同じ働きをします。

入力中にファイルの終端になると、変換は終了します。現在の変換指定 (`%n` を除く) に一致するバイト (先行空白を使用している場合はこれを除く) が読み取られる前にファイルが終わると、現在の変換指定の実行は入力エラーで終了します。それ以外の場合は、現在の変換指定の実行がマッチングエラーで終了しない限り、続く変換指定の実行 (もし、あれば) が入力エラーで終了します。

sscanf() の文字列の終端に到達するのは、fscanf() でファイルの終端に到達するのと同様です。

入力が衝突して終了した場合、対抗する入力が入力中で読み取られないままになります。変換指定に一致しない場合、後続の空白 (改行文字を含む) は読み取られません。リテラル一致および抑制される代入が成功したかどうかは、%n 変換指定によってのみ直接確認できます。

fscanf() および scanf() 関数は、stream に関連するファイルの st\_atime フィールドを更新用にマークすることができます。st\_atime フィールドは、fgetc(3C), fgets(3C), fread(3C), fscanf(), getc(3C), getchar(3C), gets(3C), または scanf() の実行が成功したときに、ungetc(3C) への以前の呼び出しによって与えられていないデータを返す stream を使用して更新用にマークされます。

**戻り値** 正常終了の場合、これらの関数は一致と代入が成功した入力項目数を返します。一致が実行初期段階で失敗した場合は、0 が返される場合もあります。最初の一致が失敗する前、または最初の変換が行われる前に入力が終わると、EOF が返されます。ストリームが設定されているというエラー表示で読み取りエラーが発生した場合、EOF が返されエラーを表示する errno が設定されます。

**エラー** scanf() 関数が失敗、および失敗する可能性がある場合は、fgetc(3C) または fgetwc(3C) を参照してください。

さらに、以下の条件で fscanf() は失敗することがあります。

EILSEQ 入力バイトシーケンスが有効な文字を形成していない。

EINVAL 引数が不足している。

**使用法** scanf() 関数を呼び出すアプリケーションに wint\_t または wchar\_t 型の形式のオブジェクトが存在する場合、これらのオブジェクトを定義する <wchar.h> ヘッダーも含む必要があります。

**使用例** 例 1 使用例 1

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name)
```

という呼び出しの場合、入力行は次のようになります。

```
25 54.32E-1 Hamster
```

この場合、n に値 3、i に値 25、x に値 5.432 がそれぞれ代入され、name には Hamster の文字列が入ります。

例 2 使用例 2

```
int i; float x; char name[50];
(void) scanf("%2d%f*d %[0123456789]", &i, &x, name);
```

という呼び出しの場合、入力行は次のようになります。

```
56789 0123 56a72
```

## scanf(3C)

### 例 2 使用例 2 (続き)

この場合、*i* に 56、*x* に 789.0 が代入され、0123 はスキップして、*name* には 56\0 が入ります。getchar(3C) への次の呼び出しでは a の文字を返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 fgetc(3C), fgets(3C), fgetwc(3C), fread(3C), isspace(3C), printf(3C), setlocale(3C), stdarg(3HEAD), strtod(3C), strtol(3C), strtoul(3C), wcrctomb(3C), ungetc(3C), attributes(5)

名前 | setlocale – プログラムのロケールの変更および照会

形式 | #include <locale.h>

```
char *setlocale(int category, const char *locale);
```

機能説明 | setlocale() 関数は、*category* および *locale* の各引数によって、指定されたプログラムのロケールの適切な部分を選択します。*category* のとる値は、LC\_CTYPE、LC\_NUMERIC、LC\_TIME、LC\_COLLATE、LC\_MONETARY、LC\_MESSAGES、および LC\_ALL です。これらの名前は <locale.h> ヘッダーに定義されています。変数 LC\_ALL はプログラムのロケールカテゴリ全体を指定します。

変数 LC\_CTYPE は、isdigit(3C) や tolower(3C) などの文字処理関数、および mbtowl(3C) や wctomb(3C) などの複数バイト文字関数の動作に影響を与えます。

変数 LC\_NUMERIC は、書式付き入出力関数と文字列変換関数用の浮動小数点形式の数字や数多くある区切り文字を左右します。

変数 LC\_TIME は、ascftime(3C)、cftime(3C)、getdate(3C)、strftime(3C)、および strptime(3C) によって渡される日付と時間の書式を左右します。

変数 LC\_COLLATE は strcoll(3C) および strxfrm(3C) のような照合関数によって生じるソート順を左右します。

変数 LC\_MONETARY は、localeconv(3C) によって返される通貨用書式の情報を左右します。

変数 LC\_MESSAGES は dgettext(3C)、gettext(3C)、および gettext(3C) のようなメッセージ関数の動作を左右します。

*locale* の値 *C* は 今までの UNIX システムの動作を指定します。プログラムの起動時には、以下が実行されます。これは環境 *C* で記述されたロケールに各カテゴリを初期化することを意味します。

```
setlocale(LC_ALL, "C")
```

*locale* の値 "" は、ロケールを環境変数から取り込まなければならないことを指定します。各種カテゴリについて環境変数を検査する順序を以下に示します。

カテゴリ	第 1 の環境変数	第 2 の環境変数	第 3 の環境変数
LC_CTYPE:	LC_ALL	LC_CTYPE	LANG
LC_COLLATE:	LC_ALL	LC_COLLATE	LANG
LC_TIME:	LC_ALL	LC_TIME	LANG
LC_NUMERIC:	LC_ALL	LC_NUMERIC	LANG
LC_MONETARY:	LC_ALL	LC_MONETARY	LANG

## setlocale(3C)

カテゴリ	第 1 の環境変数	第 2 の環境変数	第 3 の環境変数
LC_MESSAGES:	LC_ALL	LC_MESSAGES	LANG

*locale* に文字列に対するポインタが指定されていると、`setlocale()` は、指定したカテゴリのロケールを *locale* に設定しようとします。`setlocale()` が成功すると *locale* が返されます。`setlocale()` が失敗すると NULL ポインタが返され、プログラムのロケールは変更されません。

LC\_ALL カテゴリについては、その動作は少し異なります。*locale* に文字列に対するポインタが指定され、*category* に LC\_ALL が指定されている場合、`setlocale()` はすべてのカテゴリのロケールを *locale* に設定しようとします。*locale* は単一のロケールで構成された単純なロケールである場合と、複合のロケールで構成された複合ロケールである場合があります。ロケール変更を試みたあと、すべてのカテゴリに対するロケールが同じである場合、`setlocale()` は共通なロケール名へのポインタを返します。カテゴリ中にロケールが混合して存在する場合、`setlocale()` は、それぞれのロケール名を合成したものを返します。

戻り値 正常終了の場合、`setlocale()` は、新しいロケールに指定したカテゴリに関連する文字列を返します。正常終了でない場合、`setlocale()` は NULL ポインタを返します。プログラムのロケールは変更されません。

*locale* が NULL ポインタであると、`setlocale()` は、プログラムの現在のロケールの *category* に関連する文字列のポインタを返します。プログラムのロケールは変更されません。

`setlocale()` が返す文字列は、プログラムのロケール部分を復元する、文字列や関連 *category* 付きの後続の呼び出しなどです。返される文字列はプログラムによって修正してはいけません。ただし、`setlocale()` への後続の呼び出しによって、上書きされる可能性があります。

エラー 定義されているエラーはありません。

ファイル `/usr/lib/locale/locale` *locale* のロケールデータベースディレクトリ

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `locale(1)`, `ctype(3C)`, `getdate(3C)`, `gettext(3C)`, `gettext(3C)`, `isdigit(3C)`, `localeconv(3C)`, `mbtowc(3C)`, `strcoll(3C)`, `strftime(3C)`, `strptime(3C)`, `strxfrm(3C)`, `tolower(3C)`, `wctomb(3C)`, `libc(3LIB)`, `attributes(5)`, `environ(5)`, `locale(5)`

注意事項 マルチスレッドアプリケーションでロケールを変更するには、ロケールに依存するルーチンを使う前に `setlocale()` を呼び出してください。現在のロケールを知る目的で `setlocale()` を使用することはこの範囲においては可能で、マルチスレッドアプリケーションのどこでも使うことができます。

種々のロケールカテゴリが混ざり合っている場合、それらに互換性があるかどうかを確認するのはユーザーの責任です。たとえば、`LC_CTYPE=C` および `LC_TIME=ja` (`ja` とは Japanese のこと) を設定することはできません。日本時間は C ロケールの ASCII コードセットで表示できないからです。

`setlocale()` の国際化機能は、`libc` の動的リンクバージョンが、アプリケーションとリンクしている場合にのみサポートされます。`libc` の静的リンクバージョンが、アプリケーションとリンクしている場合は、`setlocale()` は C ロケールと POSIX ロケールだけを扱うことができます。

## snprintf(3C)

名前	printf, fprintf, sprintf, snprintf – 書式付き出力
形式	<pre>#include &lt;stdio.h&gt;  int printf(const char *format, /* args*/ ...); int fprintf(FILE *stream, const char *format, /* args*/ ...); int sprintf(char *s, const char *format, /* args*/ ...); int snprintf(char *s, size_t n, const char *format, /* args*/ ...);</pre>
機能説明	<p>printf() 関数は、標準出力ストリーム stdout 上に出力します。</p> <p>fprintf() 関数は、出力ストリーム <i>stream</i> 上に出力します。</p> <p>sprintf() 関数は、<i>s</i> から始まる連続したバイトを出力し、最後に NULL バイト (\0) をつけます。ユーザーは、十分な記憶領域が利用できることを確認する必要があります。</p> <p>snprintf() 関数は、sprintf() に引数 <i>n</i> を追加したのと同様です。<i>n</i> には <i>s</i> によって参照されるバッファサイズが指定されます。バッファは、常に NULL バイトで終了します。</p> <p>これらの各関数は、<i>format</i> の制御下で各引数の変換、書式化および出力を行います。<i>format</i> とは、初期シフトの状態 (もしあれば) で始まるか、または終わる文字列のことです。<i>format</i> は、以下に示す 0 個以上の指示語で構成されます。</p> <ul style="list-style-type: none"><li>■ 通常文字。出力ストリームに単純にコピーされる</li><li>■ 変換指定。各指定の結果として 0 個以上の引数を取り出す</li></ul> <p><i>format</i> 引数が不十分である場合、結果は未定義です。引数が残っていて、<i>format</i> が使い果たされた場合、余分な引数は評価されますが、評価されたとしても無効になります。</p> <p>変換は、引数リストの <i>format</i> のあとに続く、次の未使用の引数ではなく、<i>n</i> 番目の引数に適用されます。この場合、変換文字 % (後述を参照) は %<i>n</i>\$ シーケンスに置換されます。ここで <i>n</i> は [1, NL_ARGMAX] の範囲の 10 進数の整数で、引数リストにある引数の位置を示します。この機能によって、特定の言語に適切な順序で引数を選択する、書式文字列を定義できます (「使用例」参照)。</p> <p>書式文字列に、変換指定の書式 %<i>n</i>\$ が含まれる場合、引数リスト中の番号付けされた引数は、要求された回数だけ書式文字列から参照されます。</p> <p>書式文字列に、変換指定の書式 % が含まれる場合、引数リスト中の各引数は 1 度だけ使用されます。</p> <p>printf() 関数のすべての形式において、言語依存の小数点文字を出力文字列に挿入できます。小数点文字は、プログラムのロケール (LC_NUMERIC カテゴリ) によって定義されます。POSIX ロケールおよび小数点文字が定義されていないロケールでは、小数点文字のデフォルトはピリオド (.) です。</p>



- 変換指定 各変換指定は % 文字、または %n\$ 文字シーケンスによって導入されます。これらの文字の後には、以下に示す (順番で) 文字列が続きます。
- 変換すべき次の引数を指定する、最後に \$ が付いた 10 進数の文字列のフィールド (任意)。このフィールドがない場合は、最後に変換された引数の次の *args* が変換されます。
  - 変換指定の意味を変更する 0 個以上の *flags* (順不同)。
  - *field width* の最小値 (任意)。変換された値がフィールド幅より少ないバイトである場合は、デフォルトとしてフィールド幅の左側に (以下で説明する左位置合わせフラグ (-) が指定されている場合は右側に)、空白をパディング (文字詰め) します。フィールド幅は、アスタリスク (\*) (以下に説明) または 10 進数の整数の形式を取ります。  
変換文字が s の場合、標準準拠のアプリケーション (standards(5) を参照) は、フィールド幅を、出力される場合の最小バイト数として解釈します。また、標準準拠ではないアプリケーションは、フィールド幅を、画面表示の最小カラム数として解釈します。標準準拠ではないアプリケーションに対しては、たとえば %10s とは、変換された値が 7 カラムの画面幅の場合に右側に 3 個分の空白がパディングされることを意味します。  
書式が %ws ならば、フィールド幅を画面表示の最小カラム数として解釈します。
  - 精度を指定する *precision* (任意)。d、i、o、u、x、および X 変換の場合、数値の最小桁数を表します。この場合、このフィールドの先頭は 0 でパディングされます。e、E、および f 変換の場合、小数点文字以下の数字の桁数を表します。g および G 変換の場合、最大有効桁数を表します。また、s および S 変換の文字列からは、最大バイト数が出力されます。精度は、最初にピリオド (.) が来て、そのあとにアスタリスク (\*) (以下に説明) または 10 進数の文字列 (任意) が続く形式になります。NULL の 10 進数の文字列は 0 として扱われます。精度がその他の変換文字で表される場合、動作は未定義です。  
変換文字が s または S の場合、標準準拠のアプリケーション (standards(5) を参照) は、精度を、出力される場合の最大バイト数として解釈します。また、標準準拠ではないアプリケーションは、精度を、画面表示の最大カラム数として解釈します。標準準拠ではないアプリケーションに対しては、たとえば %.5s は、画面の 5 カラムに表示する文字列部分だけを出力します。完全な文字だけが書き込まれません。  
%ws の場合、精度を画面表示の最大カラム数として解釈します。精度は、最初にピリオドが来て、その後 10 進数の文字列が続く形式になります。NULL の 10 進数の文字列は 0 として扱われます。精度で指定したパディングは、フィールド幅で指定したパディングより優先されます。
  - h (任意)。変換文字 d、i、o、u、x、および X が short int 型または unsigned short int 型の引数に適用されることを指定します (引数は、整数変換 (インテグラルプロモーション) の規則に従って変換されます。値は short int または unsigned short int の形式に変換されてから表示されます)。また h は、変換文字 n が short int 型の引数へのポインタに適用されることを指定します。l (小文字のエル) (任意指定) は、変換文字 d、i、o、u、x、および X が long int 型または unsigned long int 型の引数に適用されることを指定します。また l (小文字のエル) は、変換文字 n が long int 型の引数へのポインタに適用されることを指定します。ll (小文字のエルが 2 つ) (任意指定) は、変換文字

## snprintf(3C)

d、i、o、u、x、および X が long long 型または unsigned long long 型の引数に適用されることを指定します。また ll (小文字のエルが2つ) は、変換文字 n が long long 型の引数へのポインタに適用されることを指定します。L (任意) は、変換文字 e、E、f、g、および G が long double 型の引数に適用されることを指定します。h、l、および L が上記以外の変換文字に現われたときの変換結果は、未定義です。

- l (小文字のエル) (任意)。変換文字 c が wint\_t 引数に適用されることを指定します。また L (小文字のエル) は、変換文字 s が wint\_t 引数へのポインタに適用されることを指定します。
- 適用される変換の型を意味する *conversion character* (下記参照)。

フィールド幅および精度の両方またはどちらか一方は、アスタリスク (\*) で示すこともできます。この場合、int 型の引数によってフィールド幅または精度を指定します。フィールド幅および精度の両方またはどちらか一方を指定する引数 (もしあれば) は、変換される順番で引数の前に現れなければなりません。フィールド幅が負の値の場合、- フラグの後に正のフィールド幅を指定したように扱われます。精度が負の値の場合は、精度を省略したように扱われます。%n\$ という形式で変換を指定している書式文字列では、フィールド幅または精度は \*m\$ シーケンスで示すことができます。この場合 m は、[1, NL\_ARGMAX] の範囲の 10 進数の整数で、整数引数の引数リスト内 (書式引数に続く) のフィールド幅または精度の位置を示します。以下に例を示します。

```
printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

*format* には、番号付けされた引数指定 (つまり %n\$ と \*m\$)、または番号付けされていない引数指定 (つまり % と \*) のいずれかを含むことができます。通常、両方を指定することはできません。唯一の例外として、%% と %n\$ の形式の両方を含むことがあります。*format* 文字列に、番号付けされた引数と番号付けされていない引数とを、同時に指定した場合の結果は未定義です。番号付けされた引数を指定するとき、N 番目の引数を指定する場合は、最初から (N-1) 番目までの先行引数を書式文字列内で指定する必要があります。

フラグ文字 以下に、フラグ文字とその意味を説明します。

- |           |   |
|-----------|---|
| '         | 10 進数変換 (%i、%d、%u、%f、%g、または %G) の結果の整数部分が、千単位のグループ化文字でフォーマットされます。その他の変換についての動作結果は未定義です。貨幣以外のグループ化文字が使用されます。     |
| -         | 変換の結果は、フィールド内で左詰めされます。このフラグを指定しない場合、変換の結果は右詰めされます。  |
| +         | 符号つき変換の結果は、常に符号 (+ または -) で始まります。このフラグを指定しない場合、負の値を変換するときだけ符号で始まります。  |
| 空白 (スペース) | 符号つき変換の最初の文字が符号でない場合または変換の結果に文字がない場合は、結果の前に空白がおかれます。つまり、空白のフラグと + フラグを共に指定した場合、空白のフラグは無視されます。                   |
| #         | 値は代替形式に変換されます。c、d、i、s、および u 変換の場合、このフラグは影響しません。o 変換の場合、このフラグは (必要な場合には) 精度をあげて、結果の最初の数字をゼロにします。x (または X) 変換では、0 |

以外の結果の前に 0x (または 0X) が追加されます。e、E、f、g、および G 変換の場合、必ず、結果は小数点付き数字になります。小数点の後に数字が続かない場合でも小数点が付きます (このフラグが指定されていない場合、これらの変換の結果では、小数点以降の数字がある場合のみ小数点がつけられます)。g および G 変換の場合、結果から後続 0 が削除されません (通常は削除されます)。

0 d、i、o、u、x、X、e、E、f、g、および G 変換の場合、先行 0 を (符号または基数の後ろに) 用いて、フィールド幅をパディングします。空白がパディングされることはありません。0 フラグおよび - フラグをともに指定すると、0 フラグが無視されます。d、i、o、u、x、および X の各変換では、精度を指定すると、0 フラグは無視されます。0 フラグおよび ' フラグをともに指定すると、0 パディングの前に文字が挿入されます。他の変換では、このフラグの動作は未定義です。

## 変換文字

各変換文字を指定すると、0 個以上の引数を取り出されます。書式に関する引数の指定が不十分な場合の変換結果は、未定義です。書式が終了しても引数が残っている場合は、残りの引数は無視されます。

変換文字とその意味を以下で説明します。

- d,i int 引数は、[-]dddd の形式で符号付き 10 進数に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。
- o unsigned int 引数は、dddd の形式で符号なし 8 進数の書式に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。
- u unsigned int 引数は、dddd の形式で符号なし 10 進数の書式に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。
- x unsigned int 引数は、dddd の形式で符号なし 16 進数の書式に変換されます。abcdef 文字が使用されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表せる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。
- X ABCDEF 文字が abcdef 文字の代わりに使用される点を除いて、x 変換文字と同じ動作です。
- f double の引数は、[-]ddd.ddd の形式で 10 進数に変換されます。小数点付き数字 setlocale(3C) 参照) の後の桁数は、精度で指定した数です。精度が指定されていない場合は、6 とみなされます。精度を明示的に 0 に指定

## snprintf(3C)

	<p>し、# フラグを指定しない場合、小数点付き数字は出力されません。小数点付き数字が出力されると、1 個以上の数字がその前に付きます。値は適切な桁数に丸められます。</p>
e,E	<p>double の引数は、[-]d.ddde±dd の形式に変換されます。小数点付き数字の前には 1 個の数字が付きます (引数が 0 でない場合はこの数字も 0 ではありません)。小数点付き数字の後には、精度で指定した数の数字が続きます。精度を指定しない場合は、6 とみなされます。精度が 0 で、# フラグを指定しない場合、小数点付き数字は出力されません。E 変換文字を使用すると、e ではなく E を数字につけて指数を示します。指数は、必ず 2 桁以上の数字です。値は適切な桁数に丸められます。</p>
g,G	<p>double の引数は、f または e の形式 (G 変換文字の場合は E 形式) で出力されます。精度は有効桁数を示します。明示的な精度が 0 である場合は、有効桁数が 1 桁になります。出力形式は変換される値によって決まります。変換の結果出力される指数が -4 より小さい場合、または精度以上の場合だけ、e (または E) 形式で出力されます。結果の小数部分からは後続 0 が取り除かれます。小数点付き数字は、その後に数字が続く場合だけ出力されます。</p>
c	<p>int の引数は、unsigned char 符号なし文字に変換され、結果のバイトが出力されます。</p> <p>l (小文字のエル) を修飾子として指定した場合、wint_t 引数は、精度は付けなくて、wchar_t 型の 2 つの要素の配列をポイントする引数を付けた ls 変換指定を行なった場合と同じように変換されます。最初の要素には ls 変換指定への wint_t 引数を含み、2 番目の要素には NULL ワイド文字を含みます。</p>
C	<p>lc と同じです。</p>
wc	<p>int の引数はワイド文字 (wchar_t) に変換されて出力されます。</p>
s	<p>引数は char 配列へのポインタにする必要があります。配列の中身が最後の NULL バイトまで書き込まれます (NULL バイトは含まない)。精度を指定すると、標準準拠のアプリケーション (standards(5) を参照) は、精度によって指定されるバイト数だけを書き込みます。また、標準準拠ではないアプリケーションは、精度によって指定された表示画面の カラム数に表示される文字列部分だけを出力します。精度を指定しないと場合は、無限とみなされ、最初の NULL バイトまでのすべてのバイトが出力されます。値が NULL の引数の変換は、未定義の結果になります。</p> <p>修飾子として l (小文字のエル) を指定した場合、引数は wchar_t 型の並びへのポインタである必要があります。その並びからのワイド文字コードは、NULL ワイド文字コードに至るまで、末尾に NULL ワイド文字コードを含まない文字に変換されます (最初のワイド文字が変換される前に、mbstate_t オブジェクトを 0 に初期化して記述された変換状態の、wrtomb(3C) 関数への呼び出しによる結果と同じように変換されます)。結果として生じる文字は、末尾の NULL 文字 (バイト) に至るまで (ただし、末尾には含まない) で書き込まれます。精度を指定しない場合、</p>

配列には、NULL ワイド文字を含む必要があります。精度を指定した場合、指定した文字 (バイト) 以上は書き込まれません (もしあれば、シフトシーケンスを含む)。配列が NULL ワイド文字を含む必要がある場合は、精度によって指定された文字シーケンスの長さを書き込み、関数は、一度配列の最後を過ぎたワイド文字へはアクセスを必要としません。文字の一部を書き込むことはありません。

s ls と同じです。

ws 引数は `wchar_t` 配列へのポインタの文字列にする必要があります。その配列の中身が最後の NULL 文字まで書き込まれます (NULL 文字は含まない)。精度を指定すると、精度によって指定された表示画面の列数に表示するワイド文字列部分だけを出力します。精度を指定しない場合は無限とみなされ、最初の NULL 文字までのすべてのワイド文字が出力されます。値が NULL の引数を指定した場合の変換は、未定義の結果になります。

p 引数は `void` へのポインタにする必要があります。ポインタの値は、出力可能なシーケンスに変換されます。これらの文字は、`scanf(3C)` 関数の `%p` 変換で一致した文字と同じ文字である必要があります。

n 引数は、`printf()` 関数のうちの 1 つの呼び出しにおいて、この変換文字が指定されるまでに、出力先に書き込まれたバイト数が格納される整数へのポインタです。引数は変換されません。

% % を出力します。引数は変換されません。全体の変換を指定するには `%%` にしてください。

変換指定が上記の形式のいずれにも当てはまらない場合、変換の結果は未定義となります。

浮動小数点値によって「無限」を内部的に表現している場合は、`[±]Infinity` が出力されます。希望した出力文字列の長さに応じて、`Infinity` は `Infinity` または `Inf` のどちらかになります。符号は、前述の規則に従って出力されます。

浮動小数点値によって「数でない値」を内部的に表現している場合は、`[±]NaN` が出力されます。符号は、前述の規則に従って出力されます。

フィールド幅が存在していなかったり、あるいはその値が小さい場合でも、フィールドが切り捨てられることはありません。変換の結果がフィールド幅よりも広い場合は、その結果を収容できるようにフィールドが拡張されます。`printf()` および `fprintf()` によって生成される文字は、`putc(3C)` 関数を呼び出した場合と同じように出力されます。

ファイルの `st_ctime` と `st_mtime` フィールドは、`printf()` または `fprintf()` の正常実行への呼び出しと、同じストリーム上の `fflush(3C)` または `fclose(3C)` あるいは `exit(3C)` または `abort(3C)` への呼び出しへの次の正常終了の呼び出しとの間で、更新するためにマークされます。

戻り値 `printf()`、`fprintf()`、および `sprintf()` 関数は、転送されるバイト数 (`sprintf()` の場合は最後の NULL バイトを除く) を返します。

## snprintf(3C)

	<p>snprintf() 関数はフォーマットされた文字数、つまり (バッファに十分な大きさがある場合に) バッファに書き込まれた文字数を返します。snprintf() への呼び出し時に <i>n</i> の値が 0 である場合、1 未満の指定されていない値が返されます。</p> <p>出力エラーが検出された場合、各関数は負の値を返します。</p>
エラー	<p>printf() と fprintf() の両関数が異常終了する、または異常終了する可能性がある条件については、fputc(3C) または fputwc(3C) を参照してください。</p> <p>さらに printf() のすべての形式において、以下の条件で異常終了します。</p> <p>EILSEQ            有効な文字に対応しないワイド文字コードが検出された。</p> <p>EINVAL            引数が足りない</p> <p>さらに printf() と fprintf() は、以下の条件で異常終了します。</p> <p>ENOMEM            使用可能な記憶領域が不足している</p>
使用法	<p>printf() 関数の呼び出しに wint_t または wchar_t 型のオブジェクトが存在する場合、これらのオブジェクトを定義するヘッダー &lt;wchar.h&gt; も含む必要があります。</p> <p>sprintf() と snprintf() の両関数はマルチスレッドアプリケーションでは MT-Safe です。ロケール変更のために setlocale(3C) が呼び出されていない限り、マルチスレッドアプリケーションにおいて printf() および fprintf() 関数を安全に使用できます。</p>
エスケープシーケンス	<p>通常 printf() 関数に対する書式化文字列を入力するときには、C 言語に組み込まれている以下のエスケープシーケンスを使用します。ただし、これらのエスケープシーケンスは、printf() 関数によってではなく C コンパイラによって処理されます。</p> <p>\a            ベルを鳴らして警報を出します。</p> <p>\b            バックスペース。現在位置が行頭でなければ、出力位置を現在位置の 1 文字前に移動させます。</p> <p>\f            フォームフィード。出力位置を次の論理ページの最初の出力位置に移動させます。</p> <p>\n            改行。出力位置を次の行の行頭に移動させます。</p> <p>\r            キャリッジリターン。出力位置を現在行の行頭に移動させます。</p> <p>\t            水平タブ。出力位置を、現在行上の次の水平ハードタブ位置に移動させます。</p> <p>\v            垂直タブ。出力位置を、垂直ハードタブ位置の始めに移動させます。</p> <p>また C 言語では、次の形式の文字シーケンスがサポートされています。</p> <p>\octal-number および</p>

\hex-number8 進数 octal-number または 16 進数 hex-number で表す文字に変換します。たとえば、ASCII 表現の場合、文字 a は \141 として書き込まれ、文字 Z は \132 として書き込まれます。この文法は、NULL 文字 \0 を表す場合に最もよく使用されます。これは定数ゼロ (0) とまったく同等です。通常の 8 進定数と同様に、8 進数に接頭辞ゼロが含まれていないことに注意してください。16 進数を指定するには、接頭辞の 0 を取り除いて接頭辞が x (小文字) になるようにします (大文字の X は不可です)。16 進シーケンスのサポートは、ANSI の拡張機能です。standards(5) を参照してください。

**使用例** 例 1 言語に依存する日付と時刻の形式を出力するには、以下のステートメントを使用します。

```
printf (format, weekday, month, day, hour, min);
```

米国の使用法では、*format* は次の文字列へのポインタになります。

```
"%s, %s %d, %d:%.2d\n"
```

生成されるメッセージは次のとおりです。

```
Sunday, July 3, 10:02
```

ドイツの使用法では、*format* は次の文字列へのポインタになります。

```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

生成されるメッセージは次のとおりです。

```
Sonntag, 3. Juli, 10:02
```

例 2 `Sunday, July 3, 10:02` の形式で日付と時刻を出力し、`weekday` と `month` が NULL で終わる文字列へのポインタにする場合は以下ようになります。

```
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);
```

例 3 小数点以下 5 桁まで出力する場合は以下ようになります。

```
printf("pi = %.5f", 4 * atan(1.0));
```

**デフォルト** 例 4 標準準拠ではないアプリケーション (standards(5) を参照) での `printf()` の動作だけに適用します。20 文字幅で名前のリストを表示する場合は以下ようになります。

```
printf("%20s%20s%20s", lastname, firstname, middlename);
```

**属性** 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

## snprintf(3C)

関連項目 `exit(2)`, `lseek(2)`, `write(2)`, `abort(3C)`, `ecvt(3C)`, `exit(3C)`, `fclose(3C)`,  
`fflush(3C)`, `fputwc(3C)`, `putc(3C)`, `scanf(3C)`, `setlocale(3C)`, `stdio(3C)`,  
`wcstombs(3C)`, `wctomb(3C)`, `attributes(5)`, `environ(5)`, `standards(5)`



名前 printf, fprintf, sprintf, snprintf – 書式付き出力

形式

```
#include <stdio.h>

int printf(const char *format, /* args*/ ...);
int fprintf(FILE *stream, const char *format, /* args*/ ...);
int sprintf(char *s, const char *format, /* args*/ ...);
int snprintf(char *s, size_t n, const char *format, /* args*/ ...);
```

機能説明 printf() 関数は、標準出力ストリーム stdout 上に出力します。

fprintf() 関数は、出力ストリーム *stream* 上に出力します。

sprintf() 関数は、*s* から始まる連続したバイトを出力し、最後に NULL バイト (\0) をつけます。ユーザーは、十分な記憶領域が利用できることを確認する必要があります。

snprintf() 関数は、sprintf() に引数 *n* を追加したのと同様です。*n* には *s* によって参照されるバッファサイズが指定されます。バッファは、常に NULL バイトで終了します。

これらの各関数は、*format* の制御下で各引数の変換、書式化および出力を行います。*format* とは、初期シフトの状態 (もしあれば) で始まるか、または終わる文字列のことです。*format* は、以下に示す 0 個以上の指示語で構成されます。

- 通常文字。出力ストリームに単純にコピーされる
- 変換指定。各指定の結果として 0 個以上の引数を取り出す

*format* 引数が不十分である場合、結果は未定義です。引数が残っていて、*format* が使い果たされた場合、余分な引数は評価されますが、評価されたとしても無効になります。

変換は、引数リストの *format* のあとに続く、次の未使用の引数にではなく、*n* 番目の引数に適用されます。この場合、変換文字 % (後述を参照) は %*n*\$ シーケンスに置換されます。ここで *n* は [1, NL\_ARGMAX] の範囲の 10 進数の整数で、引数リストにある引数の位置を示します。この機能によって、特定の言語に適切な順序で引数を選択する、書式文字列を定義できます (「使用例」参照)。

書式文字列に、変換指定の書式 %*n*\$ が含まれる場合、引数リスト中の番号付けされた引数は、要求された回数だけ書式文字列から参照されます。

書式文字列に、変換指定の書式 % が含まれる場合、引数リスト中の各引数は 1 度だけ使用されます。

printf() 関数のすべての形式において、言語依存の小数点文字を出力文字列に挿入できます。小数点文字は、プログラムのロケール (LC\_NUMERIC カテゴリ) によって定義されます。POSIX ロケールおよび小数点文字が定義されていないロケールでは、小数点文字のデフォルトはピリオド (.) です。

## sprintf(3C)

- 変換指定 各変換指定は % 文字、または %n\$ 文字シーケンスによって導入されます。これらの文字の後には、以下に示す (順番で) 文字列が続きます。
- 変換すべき次の引数を指定する、最後に \$ が付いた 10 進数の文字列のフィールド (任意)。このフィールドがない場合は、最後に変換された引数の次の args が変換されます。
  - 変換指定の意味を変更する 0 個以上の flags (順不同)。
  - field width の最小値 (任意)。変換された値がフィールド幅より少ないバイトである場合は、デフォルトとしてフィールド幅の左側に (以下で説明する左位置合わせフラグ (-) が指定されている場合は右側に)、空白をパディング (文字詰め) します。フィールド幅は、アスタリスク (\*) (以下に説明) または 10 進数の整数の形式を取ります。

変換文字が s の場合、標準準拠のアプリケーション (standards(5) を参照) は、フィールド幅を、出力される場合の最小バイト数として解釈します。また、標準準拠ではないアプリケーションは、フィールド幅を、画面表示の最小カラム数として解釈します。標準準拠ではないアプリケーションに対しては、たとえば %10s とは、変換された値が 7 カラムの画面幅の場合は右側に 3 個分の空白がパディングされることを意味します。

書式が %ws ならば、フィールド幅を画面表示の最小カラム数として解釈します。
  - 精度を指定する precision (任意)。d、i、o、u、x、および X 変換の場合、数値の最小桁数を表します。この場合、このフィールドの先頭は 0 でパディングされます。e、E、および f 変換の場合、小数点文字以下の数字の桁数を表します。g および G 変換の場合、最大有効桁数を表します。また、s および S 変換の文字列からは、最大バイト数が出力されます。精度は、最初にピリオド (.) が来て、そのあとにアスタリスク (\*) (以下に説明) または 10 進数の文字列 (任意) が続く形式になります。NULL の 10 進数の文字列は 0 として扱われます。精度がその他の変換文字で表される場合、動作は未定義です。

変換文字が s または S の場合、標準準拠のアプリケーション (standards(5) を参照) は、精度を、出力される場合の最大バイト数として解釈します。また、標準準拠ではないアプリケーションは、精度を、画面表示の最大カラム数として解釈します。標準準拠ではないアプリケーションに対しては、たとえば %.5s は、画面の 5 カラムに表示する文字列部分だけを出力します。完全な文字だけが書き込まれません。

%ws の場合、精度を画面表示の最大カラム数として解釈します。精度は、最初にピリオドが来て、その後 10 進数の文字列が続く形式になります。NULL の 10 進数の文字列は 0 として扱われます。精度で指定したパディングは、フィールド幅で指定したパディングより優先されます。
  - h (任意)。変換文字 d、i、o、u、x、および X が short int 型または unsigned short int 型の引数に適用されることを指定します (引数は、整数変換 (インテグラルプロモーション) の規則に従って変換されます。値は short int または unsigned short int の形式に変換されてから表示されます)。また h は、変換文字 n が short int 型の引数へのポインタに適用されることを指定します。l (小文字のエル) (任意指定) は、変換文字 d、i、o、u、x、および X が long int 型または unsigned long int 型の引数に適用されることを指定します。また l (小文字のエル) は、変換文字 n が long int 型の引数へのポインタに適用されることを指定します。ll (小文字のエルが 2 つ) (任意指定) は、変換文字

d、i、o、u、x、および X が long long 型または unsigned long long 型の引数に適用されることを指定します。また ll (小文字のエルが 2 つ) は、変換文字 n が long long 型の引数へのポインタに適用されることを指定します。L (任意) は、変換文字 e、E、f、g、および G が long double 型の引数に適用されることを指定します。h、l、および L が上記以外の変換文字に現われたときの変換結果は、未定義です。

- l (小文字のエル) (任意)。変換文字 c が wint\_t 引数に適用されることを指定します。また l (小文字のエル) は、変換文字 s が wint\_t 引数へのポインタに適用されることを指定します。
- 適用される変換の型を意味する *conversion character* (下記参照)。

フィールド幅および精度の両方またはどちらか一方は、アスタリスク (\*) で示すこともできます。この場合、int 型の引数によってフィールド幅または精度を指定します。フィールド幅および精度の両方またはどちらか一方を指定する引数 (もしあれば) は、変換される順番で引数の前に現れなければなりません。フィールド幅が負の値の場合、- フラグの後に正のフィールド幅を指定したように扱われます。精度が負の値の場合は、精度を省略したように扱われます。%n\$ という形式で変換を指定している書式文字列では、フィールド幅または精度は \*m\$ シーケンスで示すことができます。この場合 m は、[1, NL\_ARGMAX] の範囲の 10 進数の整数で、整数引数の引数リスト内 (書式引数に続く) のフィールド幅または精度の位置を示します。以下に例を示します。

```
printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

*format* には、番号付けされた引数指定 (つまり %n\$ と \*m\$)、または番号付けされていない引数指定 (つまり % と \*) のいずれかを含むことができます。通常、両方を指定することはできません。唯一の例外として、%% と %n\$ の形式の両方を含むことがあります。*format* 文字列に、番号付けされた引数と番号付けされていない引数とを、同時に指定した場合の結果は未定義です。番号付けされた引数を指定するとき、N 番目の引数を指定する場合は、最初から (N-1) 番目までの先行引数を書式文字列内で指定する必要があります。

フラグ文字 以下に、フラグ文字とその意味を説明します。

- ' 10 進数変換 (%i、%d、%u、%f、%g、または %G) の結果の整数部分が、千単位のグループ化文字でフォーマットされます。その他の変換についての動作結果は未定義です。貨幣以外のグループ化文字が使用されます。
- 変換の結果は、フィールド内で左詰めされます。このフラグを指定しない場合、変換の結果は右詰めされます。
- + 符号つき変換の結果は、常に符号 (+ または -) で始まります。このフラグを指定しない場合、負の値を変換するときだけ符号で始まります。
- 空白 (スペース) 符号つき変換の最初の文字が符号でない場合または変換の結果に文字がない場合は、結果の前に空白がおかれます。つまり、空白のフラグと + フラグを共に指定した場合、空白のフラグは無視されます。
- # 値は代替形式に変換されます。c、d、i、s、および u 変換の場合、このフラグは影響しません。o 変換の場合、このフラグは (必要な場合には) 精度をあげて、結果の最初の数字をゼロにします。x (または X) 変換では、0

## sprintf(3C)

	<p>以外の結果の前に 0x (または 0X) が追加されます。e、E、f、g、および G 変換の場合、必ず、結果は小数点付き数字になります。小数点の後に数字が続かない場合でも小数点が付きます (このフラグが指定されていない場合、これらの変換の結果では、小数点以降の数字がある場合のみ小数点がつけられます)。g および G 変換の場合、結果から後続 0 が削除されません (通常は削除されます)。</p>
0	<p>d, i, o, u, x, X, e, E, f, g, および G 変換の場合、先行 0 を (符号または基数の後ろに) 用いて、フィールド幅をパディングします。空白がパディングされることはありません。0 フラグおよび - フラグをともに指定すると、0 フラグが無視されます。d、i、o、u、x、および X の各変換では、精度を指定すると、0 フラグは無視されます。0 フラグおよび ' フラグをともに指定すると、0 パディングの前に文字が挿入されます。他の変換では、このフラグの動作は未定義です。</p>
変換文字	<p>各変換文字を指定すると、0 個以上の引数が取り出されます。書式に関する引数の指定が不十分な場合の変換結果は、未定義です。書式が終了しても引数が残っている場合は、残りの引数は無視されます。</p> <p>変換文字とその意味を以下で説明します。</p>
d,i	<p>int 引数は、[-]dddd の形式で符号付き 10 進数に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。</p>
o	<p>unsigned int 引数は、dddd の形式で符号なし 8 進数の書式に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。</p>
u	<p>unsigned int 引数は、dddd の形式で符号なし 10 進数の書式に変換されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表わすことができる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。</p>
x	<p>unsigned int 引数は、dddd の形式で符号なし 16 進数の書式に変換されます。abcdef 文字が使用されます。精度は、表示される最小桁数を指定します。変換している値を、指定した最小桁数より少ない桁数で表せる場合は、その値に先行 0 をつけて拡張します。デフォルトの精度は 1 です。精度に 0 を指定して値 0 を変換すると、文字は出力されません。</p>
X	<p>ABCDEF 文字が abcdef 文字の代わりに使用される点を除いて、x 変換文字と同じ動作です。</p>
f	<p>double の引数は、[-]ddd.ddd の形式で 10 進数に変換されます。小数点付き数字 setlocale(3C 参照) の後の桁数は、精度で指定した数です。精度が指定されていない場合は、6 とみなされます。精度を明示的に 0 に指定</p>

	し、# フラグを指定しない場合、小数点付き数字は出力されません。小数点付き数字が出力されると、1 個以上の数字がその前に付きます。値は適切な桁数に丸められます。
e,E	double の引数は、[-]d.ddde±dd の形式に変換されます。小数点付き数字の前には 1 個の数字が付きます (引数が 0 でない場合はこの数字も 0 ではありません)。小数点付き数字の後には、精度で指定した数の数字が続きます。精度を指定しない場合は、6 とみなされます。精度が 0 で、# フラグを指定しない場合、小数点付き数字は出力されません。E 変換文字を使用すると、e ではなく E を数字につけて指数を示します。指数は、必ず 2 桁以上の数字です。値は適切な桁数に丸められます。
g,G	double の引数は、f または e の形式 (G 変換文字の場合は E 形式) で出力されます。精度は有効桁数を示します。明示的な精度が 0 である場合は、有効桁数が 1 桁になります。出力形式は変換される値によって決まります。変換の結果出力される指数が -4 より小さい場合、または精度以上の場合だけ、e (または E) 形式で出力されます。結果の小数部分からは後続 0 が取り除かれます。小数点付き数字は、その後に数字が続く場合だけ出力されます。
c	int の引数は、unsigned char 符号なし文字に変換され、結果のバイトが出力されます。  l (小文字のエル) を修飾子として指定した場合、wint_t 引数は、精度は付けなくて、wchar_t 型の 2 つの要素の配列をポイントする引数を付けた ls 変換指定を行なった場合と同じように変換されます。最初の要素には ls 変換指定への wint_t 引数を含み、2 番目の要素には NULL ワイド文字を含みます。
C	lc と同じです。
wc	int の引数はワイド文字 (wchar_t) に変換されて出力されます。
s	引数は char 配列へのポインタにする必要があります。配列の中身が最後の NULL バイトまで書き込まれます (NULL バイトは含まない)。精度を指定すると、標準準拠のアプリケーション (standards(5) を参照) は、精度によって指定されるバイト数だけを書き込みます。また、標準準拠ではないアプリケーションは、精度によって指定された表示画面の カラム数に表示される文字列部分だけを出力します。精度を指定しないと場合は、無限とみなされ、最初の NULL バイトまでのすべてのバイトが出力されます。値が NULL の引数の変換は、未定義の結果になります。  修飾子として l (小文字のエル) を指定した場合、引数は wchar_t 型の並びへのポインタである必要があります。その並びからのワイド文字コードは、NULL ワイド文字コードに至るまで、末尾に NULL ワイド文字コードを含まない文字に変換されます (最初のワイド文字が変換される前に、mbstate_t オブジェクトを 0 に初期化して記述された変換状態の、wrtomb(3C) 関数への呼び出しによる結果と同じように変換されます)。結果として生じる文字は、末尾の NULL 文字 (バイト) に至るまで (ただし、末尾には含まない) で書き込まれます。精度を指定しない場合、

## sprintf(3C)

	<p>配列には、NULL ワイド文字を含む必要があります。精度を指定した場合、指定した文字 (バイト) 以上は書き込まれません (もしあれば、シフトシーケンスを含む)。配列が NULL ワイド文字を含む必要がある場合は、精度によって指定された文字シーケンスの長さを書き込み、関数は、一度配列の最後を過ぎたワイド文字へはアクセスを必要としません。文字の一部を書き込むことはありません。</p>
s	ls と同じです。
ws	引数は <code>wchar_t</code> 配列へのポインタの文字列にする必要があります。その配列の中身が最後の NULL 文字まで書き込まれます (NULL 文字は含まない)。精度を指定すると、精度によって指定された表示画面のカラム数に表示するワイド文字列部分だけを出力します。精度を指定しない場合は無限とみなされ、最初の NULL 文字までのすべてのワイド文字が出力されます。値が NULL の引数を指定した場合の変換は、未定義の結果になります。
p	引数は <code>void</code> へのポインタにする必要があります。ポインタの値は、出力可能なシーケンスに変換されます。これらの文字は、 <code>scanf(3C)</code> 関数の <code>%p</code> 変換で一致した文字と同じ文字である必要があります。
n	引数は、 <code>printf()</code> 関数のうちの 1 つの呼び出しにおいて、この変換文字が指定されるまでに、出力先に書き込まれたバイト数が格納される整数へのポインタです。引数は変換されません。
%	<code>%</code> を出力します。引数は変換されません。全体の変換を指定するには <code>%%</code> にしてください。
	<p>変換指定が上記の形式のいずれにも当てはまらない場合、変換の結果は未定義となります。</p> <p>浮動小数点値によって「無限」を内部的に表現している場合は、<code>[±]Infinity</code> が出力されます。希望した出力文字列の長さに応じて、<code>Infinity</code> は <code>Infinity</code> または <code>Inf</code> のどちらかになります。符号は、前述の規則に従って出力されます。</p> <p>浮動小数点値によって「数でない値」を内部的に表現している場合は、<code>[±]NaN</code> が出力されます。符号は、前述の規則に従って出力されます。</p> <p>フィールド幅が存在していなかったり、あるいはその値が小さい場合でも、フィールドが切り捨てられることはありません。変換の結果がフィールド幅よりも広い場合は、その結果を収容できるようにフィールドが拡張されます。<code>printf()</code> および <code>fprintf()</code> によって生成される文字は、<code>putc(3C)</code> 関数を呼び出した場合と同じように出力されます。</p> <p>ファイルの <code>st_ctime</code> と <code>st_mtime</code> フィールドは、<code>printf()</code> または <code>fprintf()</code> の正常実行への呼び出しと、同じストリーム上の <code>fflush(3C)</code> または <code>fclose(3C)</code> あるいは <code>exit(3C)</code> または <code>abort(3C)</code> への呼び出しへの次の正常終了の呼び出しとの間で、更新するためにマークされます。</p>
戻り値	<code>printf()</code> 、 <code>fprintf()</code> 、および <code>sprintf()</code> 関数は、転送されるバイト数 ( <code>sprintf()</code> の場合は最後の NULL バイトを除く) を返します。

snprintf() 関数はフォーマットされた文字数、つまり (バッファに十分な大きさがある場合に) バッファに書き込まれた文字数を返します。snprintf() への呼び出し時に *n* の値が 0 である場合、1 未満の指定されていない値が返されます。

出力エラーが検出された場合、各関数は負の値を返します。

エラー printf() と fprintf() の両関数が異常終了する、または異常終了する可能性がある条件については、fputc(3C) または fputwc(3C) を参照してください。

さらに printf() のすべての形式において、以下の条件で異常終了します。

EILSEQ 有効な文字に対応しないワイド文字コードが検出された。

EINVAL 引数が足りない

さらに printf() と fprintf() は、以下の条件で異常終了します。

ENOMEM 使用可能な記憶領域が不足している

使用法 printf() 関数の呼び出しに `wint_t` または `wchar_t` 型のオブジェクトが存在する場合、これらのオブジェクトを定義するヘッダー `<wchar.h>` も含む必要があります。

sprintf() と snprintf() の両関数はマルチスレッドアプリケーションでは MT-Safe です。ロケール変更のために `setlocale(3C)` が呼び出されていない限り、マルチスレッドアプリケーションにおいて printf() および fprintf() 関数を安全に使用できます。

エスケープシーケンス 通常 printf() 関数に対する書式化文字列を入力するときには、C 言語に組み込まれている以下のエスケープシーケンスを使用します。ただし、これらのエスケープシーケンスは、printf() 関数によってではなく C コンパイラによって処理されます。

\a ベルを鳴らして警報を出します。

\b バックスペース。現在位置が行頭でなければ、出力位置を現在位置の 1 文字前に移動させます。

\f フォームフィード。出力位置を次の論理ページの最初の出力位置に移動させます。

\n 改行。出力位置を次の行の行頭に移動させます。

\r キャリッジリターン。出力位置を現在行の行頭に移動させます。

\t 水平タブ。出力位置を、現在行上の次の水平ハードタブ位置に移動させます。

\v 垂直タブ。出力位置を、垂直ハードタブ位置の始めに移動させます。

また C 言語では、次の形式の文字シーケンスがサポートされています。

\octal-number および

## sprintf(3C)

\hex-number8 進数 octal-number または 16 進数 hex-number で表す文字に変換します。たとえば、ASCII 表現の場合、文字 a は \141 として書き込まれ、文字 Z は \132 として書き込まれます。この文法は、NULL 文字 \0 を表す場合に最もよく使用されます。これは定数ゼロ (0) とまったく同等です。通常の 8 進数と同様に、8 進数に接頭辞ゼロが含まれていないことに注意してください。16 進数を指定するには、接頭辞の 0 を取り除いて接頭辞が x (小文字) になるようにします (大文字の X は不可です)。16 進シーケンスのサポートは、ANSI の拡張機能です。standards(5) を参照してください。

**使用例** 例 1 言語に依存する日付と時刻の形式を出力するには、以下のステートメントを使用します。

```
printf (format, weekday, month, day, hour, min);
```

米国の使用法では、*format* は次の文字列へのポインタになります。

```
"%s, %s %d, %d:%.2d\n"
```

生成されるメッセージは次のとおりです。

```
Sunday, July 3, 10:02
```

ドイツの使用法では、*format* は次の文字列へのポインタになります。

```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

生成されるメッセージは次のとおりです。

```
Sonntag, 3. Juli, 10:02
```

例 2 Sunday, July 3, 10:02 の形式で日付と時刻を出力し、weekday と month が NULL で終わる文字列へのポインタにする場合は以下のようになります。

```
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);
```

例 3 小数点以下 5 桁まで出力する場合は以下のようになります。

```
printf("pi = %.5f", 4 * atan(1.0));
```

**デフォルト** 例 4 標準準拠ではないアプリケーション (standards(5) を参照) での printf() の動作だけに適用します。20 文字幅で名前のリストを表示する場合は以下のようになります。

```
printf("%20s%20s%20s", lastname, firstname, middlename);
```

**属性** 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み



sprintf(3C)

関連項目 | exit(2), lseek(2), write(2), abort(3C), ecvt(3C), exit(3C), fclose(3C),  
fflush(3C), fputwc(3C), putc(3C), scanf(3C), setlocale(3C), stdio(3C),  
wcstombs(3C), wctomb(3C), attributes(5), environ(5), standards(5)

## scanf(3C)

名前	scanf, fscanf, sscanf, vscanf, vfscanf, vsscanf – 書式付き入力の変換
形式	<pre>#include &lt;stdio.h&gt;  int <b>scanf</b>(const char *<i>format</i>, ...); int <b>fscanf</b>(FILE*<i>stream</i>, const char *<i>format</i>, ...); int <b>sscanf</b>(const char *<i>s</i>, const char *<i>format</i>, ...);  #include &lt;stdarg.h&gt; #include &lt;stdio.h&gt;  int <b>vscanf</b>(const char *<i>format</i>, va_list <i>arg</i>); int <b>vfscanf</b>(FILE *<i>stream</i>, const char *<i>format</i>, va_list <i>arg</i>); int <b>vsscanf</b>(const char *<i>s</i>, const char *<i>format</i>, va_list <i>arg</i>);</pre>
機能説明	<p>scanf() 関数は、標準入力ストリーム stdin を読み取ります。</p> <p>fscanf() 関数は、入力ストリーム <i>stream</i> を読み取ります。</p> <p>sscanf() 関数は、文字列 <i>s</i> を読み取ります。</p> <p>vscanf()、vfscanf()、および vsscanf() 関数はそれぞれ、scanf()、fscanf()、および sscanf() と同等ですが、これらの関数を呼び出すときは、可変数個の引数ではなく、&lt;stdarg.h&gt; ヘッダーで定義されている引数リスト (stdarg(3HEAD) のマニュアルページを参照) を使用します。これらの関数は va_end() マクロを呼び出しません。そのため、アプリケーションは、これらの関数を使用した後、va_end(<i>ap</i>) を呼び出してクリーンアップ作業を行う必要があります。</p> <p>これらの関数は、それぞれ、バイトを読み取り、書式にしたがってそれを解釈し、その結果を引数に格納します。これらの関数には、引数として、制御文字列 <i>format</i> (以下で説明)、および変換された入力の格納場所を示す <i>pointer</i> 群を指定します。書式に対して十分な引数が指定されていない場合、変換の結果は未定義となります。書式が終了しても引数が残っている場合、残りの引数は評価されますが、評価されたとしても無視されます。</p> <p>変換は、次の未使用の引数にではなく、引数リストの <i>format</i> のあとに続く <i>n</i> 番目の引数に適用されます。この場合、変換文字 % (後述を参照) は %<i>n</i>\$ シーケンスに置換されます。ここで <i>n</i> は [1, NL_ARGMAX] の範囲の 10 進数の整数です。この変換機能は、特定の言語に合わせた順序で引数を選択するように、書式文字列の定義を規定します。書式文字列に変換指定の書式 %<i>n</i>\$ が含まれる場合、引数リスト中の番号付けされた引数が、複数回、書式文字列から参照されるかどうかについては不定です。</p> <p><i>format</i> には、変換指定の形式、つまり % または %<i>n</i>\$ のいずれかを含むことができます。ただし、通常は 1 つの <i>format</i> 文字列に % と %<i>n</i>\$ の両方を指定することはできません。唯一の例外として、%% または %* を %<i>n</i>\$ の形式に入れることがあります。</p>

`scanf()` 関数のすべての形式において、入力文字列中の言語依存の小数点文字を検出できます。小数点文字は、プログラムのロケール (LC\_NUMERIC カテゴリ) によって定義されます。POSIX ロケール および小数点文字が定義されていないロケールでは、小数点文字のデフォルトはピリオド (.) です。

*format* とは、初期シフトの状態が始まるかまたは終わる文字列で、0 個以上の指示語 (もしあれば) で構成されます。各指示語は次のいずれか 1 つで構成されます。

- 1 つまたは複数の空白文字 (スペース、タブ、改行文字、垂直タブ、または用紙送り (form-feed) 文字)
- 通常文字 (% または空白文字は含まない)
- 変換指定

変換指定 各変換指定は % 文字、または %n\$ 文字シーケンスによって導入されます。これらの文字の後には、以下に示す (順番で) 文字列が続きます。

- 代入抑制文字 \* (任意)
- 最大フィールド幅を指定する 0 以外の 10 進数 (任意)
- 受け取るオブジェクトのサイズを表すサイズ修飾子、h、l (小文字のエル)、ll (小文字のエル 2 つ)、または L (任意)。対応する引数が `int` ではなく `short int` へのポインタである場合は、変換文字 d、i、および n の前に h を指定する必要があります。また、対応する引数が `long int` へのポインタである場合は、変換文字 d、i、および n の前に l (小文字のエル) を指定する必要があります。対応する引数が `long long int` へのポインタである場合は、変換文字 d、i、および n の前に ll (小文字のエル 2 つ) を指定する必要があります。同様に、対応する引数が `unsigned int` へのポインタではなく `unsigned short int` へのポインタである場合は、変換文字 o、u、および x の前に h を指定する必要があります。また、対応する引数が `unsigned long int` へのポインタである場合は、変換文字 o、u、および x の前に l (小文字のエル) を指定する必要があります。対応する引数が `unsigned long long int` へのポインタである場合は、変換文字 o、u、および x の前に ll (小文字のエル 2 つ) を指定する必要があります。対応する引数が `float` へのポインタではなく `double` へのポインタである場合は、変換文字 e、f、および g の前に l (小文字のエル) を指定する必要があります。または、対応する引数が `long double` へのポインタである場合は、変換文字 e、f、および g の前に L を指定する必要があります。最後に、対応する引数が文字形式へのポインタではなく `wchar_t` へのポインタである場合は、変換文字 c、s、および [ の前に l (小文字のエル) を指定する必要があります。h、l (小文字のエル)、ll (小文字のエル 2 つ)、および L が上記以外の変換文字の前に現われたとき、変換の結果は未定義となります。
- 適用される変換形式を指定する変換文字 (有効な変換文字については、後述の説明を参照してください)。

`scanf()` 関数は、形式の各指示を順番に実行します。指示にエラーがあった場合、以下に詳しく記述しているように、関数はエラーを返します。エラーは、入力エラー (入力バイトが無効) またはマッチングエラー (入力不適当) として記述されます。

## scanf(3C)

1 つまたは複数の空白文字で構成される指示は、有効な入力を読み取られなくなるまで、あるいは空白文字でないために読み取れずに残る最初のバイトまで、入力を読み取って実行します。

通常文字の指示は次のように実行されます。次のバイトが入力から読み取られ、指示を含むバイトと比較されます。比較の結果、両者が同じでない場合、指示はエラーで終了し、違いのあった後続のバイトが読み取れずに残ります。

変換指定を表す指示は、一連のマッチング入力シーケンスを定義します (各変換文字については以下に記述)。変換指定は次の手順で実行されます。

変換指定に変換文字 `l`、`c`、`C`、または `n` が含まれる場合を除いて、空白文字の入力 (`isspace(3C)` で指定) はスキップされます。

変換指定に変換文字 `n` が含まれる場合を除いて、項目は入力から読み取られます。入力項目は、入力バイトの最も長いシーケンスとして (変換文字によっては文字またはバイト単位で測定される、指定された最大フィールド幅まで) 定義されます。入力項目幅はマッチングシーケンスの最初のサブシーケンスです。入力項目に続く最初のバイト (もしあれば) は、読み取れずに残ります。入力項目の長さが 0 の場合、変換指定の実行はエラーで終了します。この状態は、ファイルの終わり、コード化エラー、あるいはストリームからの入力を妨げる読み取りエラー (この場合は入力エラー) でない限り、マッチングのエラーです。

変換文字が `%` ではない場合の入力項目 (あるいは変換指定が `%n` の場合の入力バイト数) は、変換文字に適切な形式に変換されます。入力項目がマッチングシーケンスでない場合、変換指定はエラーで終了します。この状態はマッチングエラーです。代入抑制文字が `*` で表された場合を除いて、変換の結果は、以前に変換結果を受け取っていない `format` 引数に続く最初の引数によって、ポイントされたオブジェクトに出力されます (変換指定が `%` によって呼び出されていない場合)。変換指定が文字シーケンス `%n$` によって呼び出された場合は、`n` 番目の引数に出力されます。このオブジェクトが適切な形式ではない場合、または変換の結果が提供された空間に対応できない場合、動作結果は未定義です。

変換文字 以下に、有効な変換文字を示します。

- d 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- i 整数 (符号は任意) と一致します。書式は、`strtoul()` の `base` 引数に値 0 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- o 8 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 8 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。

- u 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- x 16 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 16 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- e,f,g 符号付き浮動小数点数に一致します。書式は `strtod(3C)` の変換対象となる文字コードの並びと同じです。サイズ修飾子を指定しない場合、対応する引数は `float` へのポインタである必要があります。
- `printf(3C)` ファミリの関数が、ANSI/IEEE Std 754: 1985 標準を指定できるように、無限および NaN (浮動小数点形式にコード化された 7858 シンボリックエンティティ) 用に文字列表現を生成した場合、`scanf()` ファミリの関数はそれらを入力として認識します。
- s 空白でないバイトシーケンスに一致します。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- 1 (小文字のエル) 修飾子が与えられた場合、入力 は初期シフト状態で始まる文字シーケンスです。それぞれの文字は、`mbrtowc(3C)` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- [ 複数の文字 (*scanset* という) からなる、空でない文字シーケンスと一致します。この場合、通常行われる先行する空白のスキップは抑制されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL バイトを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- 1 (小文字のエル) 修飾子が与えられた場合、入力 は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- 変換指定は、*format* 文字列のすべての後続の文字、つまり、対応する右角括弧 (]) まで (または、その括弧も含む) が入ります。括弧内の文字 (*scanlist*) は *scanset* を構成しています。左角括弧の次の文字がサーカンフ

レックス (^) の場合は、サーカンフレックスと直角括弧の間 *scanlist* に入っていないすべての文字が *scanset* を構成しています。変換指定が [ ] または [^] で始まる場合は、この直角括弧は *scanlist* 内に含まれており、次の直角括弧が指定の終了を示す直角括弧になります。*scanlist* に - があり、その - が最初の文字ではない場合、最初の文字が ^ で - が 2 番目の文字ではない場合、あるいは - が最後の文字ではない場合、一致する文字の範囲が表示されます。

**c** フィールド幅に指定された数 (フィールド幅が変換指定にない場合は 1) の文字シーケンスと一致します。対応する引数は、その文字シーケンスを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。NULL 文字は追加されません。この場合、通常行われる先行する空白のスキップは抑制されます。

l (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよび最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。NULL ワイド文字は追加されません。

**p** 対応する `printf(3C)` 関数の `%p` 変換によって出力されるシーケンスと同様のシーケンスと一致します。対応する引数は `void` へのポインタである必要があります。入力項目が、同一プログラムの実行中で以前に変換した値である場合は、ポインタは先にその値を変換したときと同様の比較を行います。そうでない場合は、`%p` 変換の動作は未定義です。

**n** 入力は使用 (変換) されません。対応する引数は、これまでに `scanf()` 関数を呼び出して入力ストリームから読み取ったバイト数を示す整数へのポインタである必要があります。変換指定 `%n` を実行しても、この関数の実行終了時に返される代入数は増加しません。

**C** lC と同じです。

**S** lS と同じです。

**%** 単一の % と一致します。変換または代入は行われません。完全な変換指定をするには %% を指定してください。

変換指定が正しくない場合の動作は未定義です。

変換文字 E、G、および X はそれぞれ e、g、および x と同じ働きをします。

入力中にファイルの終端になると、変換は終了します。現在の変換指定 (`%n` を除く) に一致するバイト (先行空白を使用している場合はこれを除く) が読み取られる前にファイルが終わると、現在の変換指定の実行は入力エラーで終了します。それ以外の場合は、現在の変換指定の実行がマッチングエラーで終了しない限り、続く変換指定の実行 (もし、あれば) が入力エラーで終了します。

sscanf() の文字列の終端に到達するのは、fscanf() でファイルの終端に到達するのと同様です。

入力が衝突して終了した場合、対抗する入力が入力中で読み取られないままになります。変換指定に一致しない場合、後続の空白 (改行文字を含む) は読み取られません。リテラル一致および抑制される代入が成功したかどうかは、%n 変換指定によってのみ直接確認できます。

fscanf() および scanf() 関数は、stream に関連するファイルの st\_atime フィールドを更新用にマークすることができます。st\_atime フィールドは、fgetc(3C), fgets(3C), fread(3C), fscanf(), getc(3C), getchar(3C), gets(3C), または scanf() の実行が成功したときに、ungetc(3C) への以前の呼び出しによって与えられていないデータを返す stream を使用して更新用にマークされます。

**戻り値** 正常終了の場合、これらの関数は一致と代入が成功した入力項目数を返します。一致が実行初期段階で失敗した場合は、0 が返される場合もあります。最初の一致が失敗する前、または最初の変換が行われる前に入力が終わると、EOF が返されます。ストリームが設定されているというエラー表示で読み取りエラーが発生した場合、EOF が返されエラーを表示する errno が設定されます。

**エラー** scanf() 関数が失敗、および失敗する可能性がある場合は、fgetc(3C) または fgetwc(3C) を参照してください。

さらに、以下の条件で fscanf() は失敗することがあります。

EILSEQ 入力バイトシーケンスが有効な文字を形成していない。

EINVAL 引数が不足している。

**使用法** scanf() 関数を呼び出すアプリケーションに wint\_t または wchar\_t 型の形式のオブジェクトが存在する場合、これらのオブジェクトを定義する <wchar.h> ヘッダーも含む必要があります。

**使用例** 例 1 使用例 1

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name)
```

という呼び出しの場合、入力行は次のようになります。

```
25 54.32E-1 Hamster
```

この場合、n に値 3、i に値 25、x に値 5.432 がそれぞれ代入され、name には Hamster の文字列が入ります。

例 2 使用例 2

```
int i; float x; char name[50];
(void) scanf("%2d%f*d %[0123456789]", &i, &x, name);
```

という呼び出しの場合、入力行は次のようになります。

```
56789 0123 56a72
```

## sscanf(3C)

### 例 2 使用例 2 (続き)

この場合、*i* に 56、*x* に 789.0 が代入され、0123 はスキップして、*name* には 56\0 が入ります。getchar(3C) への次の呼び出しでは a の文字を返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 fgetc(3C), fgets(3C), fgetwc(3C), fread(3C), isspace(3C), printf(3C), setlocale(3C), stdarg(3HEAD), strtod(3C), strtol(3C), strtoul(3C), wctomb(3C), ungetc(3C), attributes(5)



名前	strptime, cftime, ascftime – 日付と時刻を文字列に変換
形式	<pre>#include &lt;time.h&gt;  size_t <b>strptime</b>(const char *s, size_t maxsize, const char *format,                  const struct tm *timeptr);  int <b>cftime</b>(char *s, char *format, const time_t *clock);  int <b>ascftime</b>(char *s, const char *format, const struct tm *timeptr);</pre>
機能説明	<p>strptime()、ascftime()、cftime() 関数は、<i>format</i> がポイントする文字列の制御に従って、バイトを <i>s</i> がポイントする配列に格納します。<i>format</i> 文字列は、0 個以上の変換仕様と通常文字からなります。変換仕様は、パーセント記号 (%) の後に、変換の内容を表す 1 つ以上の文字を伴います。終了 NULL バイトを含む通常文字すべては、そのまま <i>s</i> が示す配列に複写されます。オーバーラップしている複数のオブジェクト間で複写が行われると、変換の結果は予測できません。strptime() の場合は、<i>maxsize</i> を超えないバイト数の文字だけが配列に格納されます。</p> <p><i>format</i> が (char *)0 であれば、そのロケールのデフォルト形式が使用されます。strptime() の場合は、デフォルト形式は %c と同じになります。また、cftime() や ascftime() の場合は、デフォルト形式は %C と同じになります。cftime() も ascftime() も、まず環境変数の値 CFTIME を利用しようとしませんが、これが未定義または空であれば、デフォルト形式を使用します。</p> <p>各変換仕様は、以下の一覧に示したように、適切な文字に置き換えられます。適切な文字は、プログラムのロケールの LC_TIME カテゴリと、strptime() と ascftime() の <i>timeptr</i> がポイントする構造体に入っている値、cftime() の <i>clock</i> で表されている時間で決まります。</p>
デフォルト規格準拠	<pre>%%      % と同じ %a       省略形式の曜日名 (ロケール固有) %A       完全形式の曜日名 (ロケール固有) %b       省略形式の月名 (ロケール固有) %B       完全形式の月名 (ロケール固有) %c       日時の表現 (ロケール固有) %C       date(1) が出力する形式の日時の表現 %C       世紀の番号。年を 100 で除算して余りを切り捨てて得られる 1 から 99 までの整数。1 桁の場合は前に 0 が付加される (standards(5) を参照) %d       日 (1 から 31)。1 桁の場合は前に 0 が付加される %D       %m/%d/%y 形式で表した日付 %e       日 (1 から 31)。1 桁の場合は前に空白文字が付加される %g       週をベースにした西暦年の下 2 桁 (00 から 99) %G       週をベースにした西暦年の完全表示 (0000 から 9999)</pre>

## strftime(3C)

%h	省略形式の月名 (ロケール固有)
%H	24 時間表示で表した時 (0 から 23)。1 桁の場合は前に 0 が付加される
%I	12 時間表示で表した時 (1 から 12)。1 桁の場合は前に 0 が付加される
%j	年の通算日 (1 から 366)。3 桁未満の場合は前に 0 が付加される
%k	24 時間表示で表した時 (0 から 23)。1 桁の場合は前に空白文字が付加される
%l	12 時間表示で表した時 (1 から 12)。1 桁の場合は前に空白文字が付加される
%m	月 (1 から 12)。1 桁の場合は前に 0 が付加される
%M	分 (00 から 59)。先行する 0 は任意
%n	復帰改行を挿入
%p	午前または午後を表す値 (ロケール固有)
%r	%p を伴った、12 時間表示形式で表した時刻
%R	%H:%M 形式で表した時刻
%S	秒 (00 から 61)。値の範囲が (00 から 59) ではなく (00 から 61) であるのは、うるう秒および二重うるう秒に対処するため
%t	タブを挿入
%T	%H:%M:%S 形式で表した時刻
%u	曜日を表す番号 (1 から 7)。日曜日が 1 (「注意事項」を参照)
%U	その年の何週目かを表す数値 (00 から 53)。第 1 週はその年の最初の日曜日 から始まるとする
%V	ISO 8601 で何週目かを表す数値 (01 から 53)。ISO 8601 の週をベースとするシステムでは、週は月曜日 から始まり、1 月 4 日およびその年の最初の木曜日の両方を含む週が第 1 週となる。その年の最初の月曜日が 1 月 1 日、2 日、3 日、または 4 日の場合には、その週は前年の最終週の一部になる (「注意事項」を参照)
%w	曜日を表す番号 (0 から 6)。日曜日が 0
%W	その年の何週目かを表す数値 (00 から 53)。第 1 週はその年の最初の月曜日 から始まるとする
%x	適切な日付の表現 (ロケール固有)
%X	適切な時間の表現 (ロケール固有)
%y	西暦年の下 2 桁 (00 から 99)
%Y	西暦年の完全表示 (たとえば 1993)
%Z	タイムゾーン名 (タイムゾーンがない場合には文字なし)

上記の変換仕様または後述する変更指定のどれにも該当しない指定を記述すると、変換結果は未定義で 0 が返されます。

%U と %W (このほか、後述の %OU と %OW) とでは、何曜日を週の第 1 日目とするかが異なっています。たとえば、1 月の第 1 週目 (週番号 1) は、%U では日曜日から、%W では月曜日から始まります。また、週番号 0 には、%U では 1 月の第 1 日曜日より前の日が、%W では 1 月の第 1 月曜日より前の日が含まれています。

#### 変換仕様の変更

上記の変換仕様に、変更を表す文字 E または O を付加することができます。このようにすると、代替形式または代替指定で値を得ることが可能です。希望した代替形式または代替指定が現ロケール中に存在していなければ、変更を示す文字を指定しなかった場合の形式で値が得られます。

%Ec	適切な日付および時刻の代替形式 (ロケール固有)
%EC	代替表示形式として指定されている年号 (ロケール固有)
%Eg	代替表示形式として指定されている年号 (%EC) に対応した週をベースとした年 (ロケール固有)
%EG	完全形式の週をベースとした代替年表示
%Ex	日付の代替表示形式 (ロケール固有)
%EX	時刻の代替表示形式 (ロケール固有)
%Ey	代替表示形式として指定されている年号 (%EC) に対応した年 (ロケール固有)
%EY	完全形式の代替年表示
%Od	日をロケール固有の代替数値記号で表す
%Oe	%Od と同じ
%Og	ロケール固有の週をベースとした代替表示の年 (%c に対応) の値を、ロケール固有の代替数値記号で表す
%OH	24 時間表示での時をロケール固有の代替数値記号で表す
%OI	12 時間表示での時をロケール固有の代替数値記号で表す
%Om	月をロケール固有の代替数値記号で表す
%OM	分をロケール固有の代替数値記号で表す
%OS	秒をロケール固有の代替数値記号で表す
%Ou	週日を示す値をロケール固有の代替数値記号にある数字で表す
%OU	その年の何週目かをロケール固有の代替数値記号で表す。週は日曜日から始まるとする
%Ow	曜日を示す値をロケール固有の代替数値記号で表す。日曜日を 0 とする
%OW	その年の何週目かをロケール固有の代替数値記号で表す。週は月曜日から始まるとする

## strftime(3C)

%Oy	ロケール固有の代替表示の年 (%c に対応) の値を、ロケール固有の代替数値記号で表す						
出力言語の選択	デフォルトでは、 <code>strftime()</code> 、 <code>cftime()</code> 、 <code>ascftime()</code> は米国英語で表示されます。 <code>setlocale()</code> で <code>LC_TIME</code> カテゴリのロケールを設定すると、 <code>strftime()</code> 、 <code>cftime()</code> 、 <code>ascftime()</code> の実行結果を特定の言語で出力できるようになります。						
タイムゾーン	<code>tzset(3C)</code> を呼び出した場合と同じタイムゾーン情報を使用します。						
戻り値	<code>strftime()</code> 、 <code>cftime()</code> 、 <code>ascftime()</code> 関数は、 <code>s</code> が示す配列に書き出した文字数を返します。これには終了 <code>NULL</code> 文字は含まれません。生成された文字の合計数 (終了 <code>NULL</code> 文字を含む) が <code>maxsize</code> を超えていると、 <code>strftime()</code> は、0 を返し、配列の内容は未確定になります。						
使用例	<p>例 1 An example of the <code>strftime()</code> function.</p> <p>ここでは、POSIX ロケールの <code>strftime()</code> の使用例を示します。<code>tmpr</code> が指す構造体に 1986 年 8 月 28 日木曜日 12 時 44 分 36 秒に対応する値が入っている場合、文字列 <code>str</code> の内容がどうなるかを示します。</p> <pre>strftime (str, strsize, "%A %b %d %j", tmpr)</pre> <p>これにより、<code>str</code> の内容は "Thursday Aug 28 240" になります。</p>						
属性	次の属性については <code>attributes(5)</code> のマニュアルページを参照してください。						
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">属性タイプ</th> <th style="text-align: center;">属性値</th> </tr> </thead> <tbody> <tr> <td>MT レベル</td> <td>MT-safe</td> </tr> <tr> <td>CSI</td> <td>対応済み</td> </tr> </tbody> </table>	属性タイプ	属性値	MT レベル	MT-safe	CSI	対応済み
属性タイプ	属性値						
MT レベル	MT-safe						
CSI	対応済み						
関連項目	<code>date(1)</code> , <code>ctime(3C)</code> , <code>mkttime(3C)</code> , <code>setlocale(3C)</code> , <code>strptime(3C)</code> , <code>tzset(3C)</code> , <code>TIMEZONE(4)</code> , <code>zoneinfo(4)</code> , <code>attributes(5)</code> , <code>environ(5)</code> , <code>standards(5)</code>						
注意事項	<p>Solaris 7 では、<code>%v</code> の変換仕様が変わりました。この変更は、現時点で公表されている ISO C9x 標準の決定に基づいています。以前の仕様では、1 月 1 日を含む週に、4 日未満の日数の新しい年が含まれている場合、その週は前年の第 53 週となりましたが、ISO C9x 標準委員会によって、この仕様が正しくないと判断されました。</p> <p><code>%g</code>、<code>%G</code>、<code>%Eg</code>、<code>%EG</code>、<code>%Og</code> の変換仕様が Solaris 7 に追加されました。この変更は、現時点で公表されている ISO C9x 標準の決定に基づいています。これらの仕様は ISO C9x 標準の決定によって変更される可能性があり、変更前の仕様との互換性が保たれないこともあります。</p> <p>Solaris 8 リリースでは、<code>%u</code> の変換仕様に変更されました。この変更は、XPG4 仕様に基づいたものです。</p>						

strftime(3C)

%Z 指示子と zoneinfo タイムゾーンを使用している場合に、入力日時が 20:45:52 UTC, December 13, 1901 から 03:14:07 UTC, January 19, 2038 までの範囲外にある場合、タイムゾーン名が正しくないことがあります。

## textdomain(3C)

名前	gettext, dgettext, dcgettext, ngettext, dngettext, dcngettext, textdomain, bindtextdomain, bind_textdomain_codeset – メッセージ処理関数
形式	
Solaris および GNU 互換	<pre>#include &lt;libintl.h&gt; char *gettext(const char *msgid); char *dgettext(const char *domainname, const char *msgid); char *textdomain(const char *domainname); char *bindtextdomain(const char *domainname, const char *dirname); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcgettext(const char *domainname, const char *msgid, int category);</pre>
GNU 互換	<pre>#include &lt;libintl.h&gt; char *ngettext(const char *msgid1, const char *msgid2, unsigned long int n); char *dngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n); char *bind_textdomain_codeset(const char *domainname, const char *codeset); #include &lt;libintl.h&gt; #include &lt;locale.h&gt; char *dcngettext(const char *domainname, const char *msgid1, const char *msgid2, unsigned long int n, int category);</pre>
機能説明	<p>gettext(), dgettext(), dcgettext() 関数は、特定のドメイン内において、また、現在のロケールにおいて、指定の <i>msgid</i> 引数に基づいたターゲット文字列を取得します。gettext(), dgettext(), dcgettext() から返される文字列の長さは、これらの関数が呼び出されるまでは未定です。<i>msgid</i> 引数は、NULL で終わる文字列です。</p> <p>ngettext(), dngettext(), および dcngettext() 関数はそれぞれ、gettext(), dgettext(), および dcgettext() と同等ですが、GNU 互換メッセージカタログだけを処理することができ、複数形の処理方法が異なります。メッセージ文字列を検索するとき、ngettext(), dngettext(), および dcngettext() 関数は <i>msgid1</i> 引数を検索鍵として使用し、<i>n</i> 引数で複数形を決定します。メッセージカタログが見つからない場合、<i>n</i> == 1 であれば <i>msgid1</i> が返され、そうでなければ <i>msgid2</i> が返されます。</p> <p>環境変数 NLSPATH (environ(5) を参照) が、最初に LC_MESSAGES カタログの位置として検索されます。現在のロケールの LC_MESSAGES カテゴリの設定によって、gettext() および dgettext() が文字列取得に使用するロケールが決まります。</p>

*category* 引数は `dcgettext()` の使用するロケールを決めます。NLSPATH が定義されておらず、現在のロケールが C ロケールの場合、`gettext()`、`gettext()`、`dcgettext()` は、渡されたメッセージ文字列をそのまま返します。C 以外のロケールでは、NLSPATH が定義されていない場合、あるいは、NLSPATH で指定したコンポーネントでメッセージカタログが見つからない場合、メッセージカタログの検索には、次の段落で説明するスキームが使用されます。

LANGUAGE 環境変数は、使用する GNU 互換メッセージカタログを決定するために調べられます。LANGUAGE の値はコロン (:) で区切られたロケール名のリストです。LANGUAGE が定義されている場合、リストに指定されている順番で各ロケールが検索されます。そして、GNU 互換メッセージカタログが見つかった場合、そのメッセージが返されます。GNU 互換メッセージが見つかったが、対応する *msgid* が見つからない場合、*msgid* 文字列が返されます。LANGUAGE が定義されていない場合、あるいは、LANGUAGE の処理中に Solaris メッセージカタログが見つかったが GNU 互換メッセージカタログが見つからない場合、*dirname/locale/category/domainname.mo* というパス名でメッセージカタログが検索されます。ここで、*dirname* は `bindtextdomain()` で指定されたディレクトリ、*locale* はロケール名、*category* は LC\_MESSAGES または LC\_XXX のどちらかです。LC\_MESSAGES は `gettext()`、`dgettext()`、`ngettext()`、または `dngettext()` を呼び出した場合に使用されます。LC\_XXX は `dcgettext()` または `dcngettext()` を呼び出した場合に使用され、XXX 部分は *category* 引数で指定したロケールカテゴリ名と同じになります。

`gettext()` と `ngettext()` の場合、使用されるドメインは、最後に行なった `textdomain()` の有効な呼び出しによって設定されます。`textdomain()` の有効な呼び出しが行われていない場合は、デフォルト時のドメイン (`messages`) が使用されます。

`dgettext()`、`dngettext()`、`dcngettext()`、および `dcgettext()` の場合、使用されるドメインは *domainname* 引数で指定されます。この *domainname* 引数は、構文と意味において、`textdomain()` の *domainname* 引数と同等ですが、このドメインの選択が `dgettext()`、`dcgettext()`、`dngettext()` および `dcngettext()` 関数の呼び出し中にのみ有効であるという点で異なります。

`textdomain()` 関数は、アクティブな LC\_MESSAGES ロケールカテゴリの現在のドメイン名の設定 または照会を行います。*domainname* 引数は、正当なファイル名に使用できる文字のみからなる、NULL で終わる文字列です。

*domainname* 引数は、システム内の、一意のドメイン名です。1つのシステム内に、同一ドメインのバージョンが複数存在する場合は、`bindtextdomain()` を使用して名前空間の衝突を回避することができます。`textdomain()` を呼び出さない場合には、デフォルト時のドメインが選択されます。最後に行なった `textdomain()` の有効な呼び出しによるドメインの設定は、このあとに行う `setlocale(3C)` および `gettext()` の呼び出しにおいても有効です。

*domainname* 引数は、現在アクティブな、LC\_MESSAGES ロケールに適用されます。

## textdomain(3C)

*domainname* に NULL ポインタを設定して `textdomain()` を呼び出すことによって、ドメインの現在の状態に影響することなく、ドメインの現在の設定を照会することができます。*domainname* 引数に NULL 文字列を設定して `textdomain()` を呼び出すと、ドメインの設定は、デフォルト時のドメイン (`messages`) になります。

`bindtextdomain()` 関数は、メッセージドメインのパス部分 *domainname* と、*dirname* に入っている値を結合します。*domainname* が空文字列ではなく、まだ結合されていない場合には、`bindtextdomain()` は、*domainname* と *dirname* を結合します。

*domainname* が空文字列ではなく、すでに結合されている場合には、`bindtextdomain()` は、事前に行われていた結合部分を *dirname* に置き換えます。*dirname* 引数に使用できるのは、`gettext()`、`dgettext()`、または `dcgettext()` が呼び出されたときに解決される、絶対パス名または相対パス名です。*domainname* が NULL ポインタまたは空文字列の場合は、`bindtextdomain()` は NULL を返します。ユーザー定義のドメイン名は、`sys_` で始めることはできません。この文字列で始まるドメイン名は、システムで使用するために予約されています。

`bind_textdomain_codeset()` 関数を使用すると、ドメイン *domainname* のメッセージカタログ用の出力コードセットを指定できます。*codeset* 引数は、`iconv_open(3C)` 関数に対して使用可能な有効なコードセット名であるか、空ポインタである必要があります。*codeset* 引数が空ポインタの場合、`bind_textdomain_codeset()` はドメイン *domainname* 用に現在選択されているコードセットを返します。コードセットが選択されていない場合、`bind_textdomain_codeset()` は空ポインタを返します。`bind_textdomain_codeset()` 関数は複数回使用できます。同じ *domainname* 引数を指定して `bind_textdomain_codeset()` 関数を複数回使用した場合、先行する呼び出しで行われた設定は後続の呼び出しによって無効にされます。`bind_textdomain_codeset()` 関数は、選択されたコードセット名を含む文字列へのポインタを返します。この文字列は `bind_textdomain_codeset()` 関数に内部的に割り当てられているため、ユーザーが変更してはなりません。

戻り値

`gettext()`、`dgettext()`、および `dcgettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*msgid* 文字列を返します。

`ngettext()`、`dngettext()`、および `dcngettext()` 関数は、検索が成功した場合、メッセージ文字列を返します。検索が失敗した場合、*n* == 1 であれば、*msgid1* を返します。そうでなければ、*msgid2* を返します。

`gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` が返す文字列の個々のバイトには、空文字以外のあらゆる値が入ります。*msgid* が NULL ポインタである場合の戻り値は未定義です。返される文字列は、プログラムで修正してはならず、この後で `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()` または `setlocale(3C)` を呼び出すことにより、無効になることがあります。`dgettext()`、`dcgettext()`、`dngettext()`、または `dcngettext()` に対する *domainname* 引数が NULL ポインタである場合、現在 `textdomain()` によってバインドされているドメインが使用されます。



## textdomain(3C)

`textdomain()` の正常な戻り値は、ドメインの現在の設定が入っている文字列へのポインタです。`domainname` が NULL ポインタの場合は、`textdomain()` は現在のドメインが入っている文字列へのポインタを返します。`textdomain()` を先に呼び出していない場合に `domainname` を空文字列にすると、デフォルト時のドメイン名が返されます。デフォルト時のドメイン名は、`messages` です。`textdomain()` が失敗した場合、NULL ポインタが返されます。

`bindtextdomain()` からの戻り値は、`dirname` が入った NULL で終わる文字列です。`dirname` が NULL の場合は、`domainname` と結び付けられたディレクトリバインディングです。何も結合されていない場合には、デフォルトの戻り値は `/usr/lib/locale` になります。`domainname` が NULL ポインタまたは空文字列の場合、`bindtextdomain()` は何も行わず、NULL ポインタを返します。返された文字列を呼び出し側で変更してはなりません。`bindtextdomain()` が失敗した場合、NULL ポインタが返されます。

使用法 これらのルーチンでは、メッセージの長さに制限はありません。ただし、テキスト `domainname` は、TEXTDOMAINMAX (256) バイト以内に制限されています。

ロケール変更のために `setlocale(3C)` が呼び出されていない限り、マルチスレッドアプリケーションにおいて `gettext()`、`dgettext()`、`dcgettext()`、`ngettext()`、`dngettext()`、`dcngettext()`、`textdomain()`、および `bindtextdomain()` 関数を安全に使うことができます。

`gettext()`、`dgettext()`、`dcgettext()`、`textdomain()`、および `bindtextdomain()` 関数は Solaris メッセージカタログと GNU 互換メッセージカタログの両方を処理できます。`ngettext()`、`dngettext()`、`dcngettext()`、および `bind_textdomain_codeset()` 関数は GNU 互換メッセージカタログだけを処理できます。Solaris メッセージカタログと GNU 互換メッセージカタログの詳細については、`msgfmt(1)` のマニュアルページを参照してください。

ファイル

`/usr/lib/locale`

メッセージ・ドメイン・ファイルのデフォルトパス部分

`/usr/lib/locale/locale/LC_MESSAGES/domainname.mo`

言語 `locale` および `domainname` 用メッセージの入っているファイルのシステムデフォルト位置

`/usr/lib/locale/locale/LC_XXX/domainname.mo`

`LC_XXX` が `LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` である場合に、`dcgettext()` 呼び出し用の言語 `locale` および `domainname` メッセージの入っているファイルのシステムデフォルト位置

`dirname/locale/LC_MESSAGES/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の、ドメイン `domainname` およびパス部分 `dirname` 用メッセージが入っているファイルの位置

`dirname/locale/LC_XXX/domainname.mo`

`bindtextdomain()` の正常な呼び出し後の `LC_XXX` が、`LC_CTYPE`、`LC_NUMERIC`、`LC_TIME`、`LC_COLLATE`、`LC_MONETARY`、または `LC_MESSAGES` のいずれかである場合 `dcgettext()` 呼び出し用のドメイン `domainname`、言語

## textdomain(3C)

属性 *locale*、およびパス部分 *dirname* メッセージが入っているファイルの位置  
次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 `msgfmt(1)`, `xgettext(1)`, `iconv_open(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

名前 toascii – 整数を 7 ビット ASCII 文字へ変換

形式 

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

機能説明 toascii() 関数は、引数を 7 ビット ASCII 文字に変換します。

戻り値 toascii() 関数は、(c & 0x7f) の値を返します。

エラー エラーは返しません。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 isascii(3C), attributes(5)

## `_tolower(3C)`

名前 `_tolower` – 大文字を小文字に変換

形式 `#include <ctype.h>`

```
int _tolower(int c);
```

機能説明 `_tolower()` マクロは、引数 `c` が大文字である必要があることを除いては `tolower(3C)` と同じです。

戻り値 正常終了時、`_tolower()` は渡された引数に対応する小文字を返します。

エラー 定義されたエラーはありません。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 `isupper(3C)`, `tolower(3C)`, `attributes(5)`

名前 tolower – 大文字を小文字に変換

形式 #include <ctype.h>

```
int tolower(int c);
```

機能説明 tolower() 関数は、int 型の値域を取ります。この範囲には、unsigned char として表現できる値と、EOF の値があります。引数が他の値である場合、引数そのまま返されます。tolower() の引数が大文字を表し、対応する小文字が存在する（プログラムのロケールカテゴリ LC\_CTYPE にある文字型情報として定義される）場合には、結果は対応する小文字になります。定義域内にあるその他すべての引数はそのまま返されます。

戻り値 正常終了時、tolower() は渡された引数に対応する小文字を返します。そうでない場合、引数をそのまま返します。

エラー 定義されたエラーはありません。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 [\\_tolower\(3C\)](#), [setlocale\(3C\)](#), [attributes\(5\)](#)

## `_toupper(3C)`

名前 `_toupper` – 小文字を大文字に変換

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

機能説明 `_toupper()` マクロは、引数 `c` が小文字である必要があることを除いては `toupper(3C)` と同じです。

戻り値 正常終了時、`_toupper()` は渡された引数に対応する大文字を返します。

エラー 定義されたエラーはありません。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 `islower(3C)`, `toupper(3C)`, `attributes(5)`

名前 toupper – 小文字を大文字に変換

形式 #include <ctype.h>

```
int toupper(int c);
```

機能説明 toupper() 関数は、int 型の値域を取ります。この範囲には、unsigned char として表現できる値と、EOF の値があります。引数が他の値である場合、引数がそのまま返されます。toupper() の引数が小文字を表し、対応する大文字が存在する (プログラムのロケールカテゴリ LC\_CTYPE にある文字型情報として定義される) 場合には、結果は対応する大文字になります。定義域内にあるその他すべての引数はそのまま返されます。

戻り値 正常終了時、toupper() は渡された引数に対応する大文字を返します。そうでない場合、引数をそのまま返します。

エラー 定義されたエラーはありません。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 \_toupper(3C), setlocale(3C), attributes(5)

## towlower(3C)

名前 towlower – ワイド文字表現の大文字を小文字に変換

形式 `#include <wchar.h>`

```
wint_t towlower(wint_t wc);
```

機能説明 towlower() 関数は、wint\_t 型の値域を取ります。この範囲は、wchar\_t として表現できる文字で、現在のロケールで有効な文字に対応するワイド文字コードの値であるか、WEOF の値である必要があります。引数が他の値である場合、引数がそのまま返されます。towlower() の引数がワイド文字コードの大文字を表し、対応するワイド文字コードの小文字が存在する (プログラムのロケールカテゴリ LC\_CTYPE にある文字型情報として定義される) 場合には、結果は対応するワイド文字コードの小文字になります。定義域内にあるその他すべての引数はそのまま返されます。

戻り値 正常終了時、towlower() は渡された引数に対応する小文字を返します。そうでない場合、引数をそのまま返します。

エラー 定義されたエラーはありません。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 iswalpna(3C), setlocale(3C), towupper(3C), attributes(5)



名前 towupper – ワイド文字表現の小文字を大文字に変換

形式 `#include <wchar.h>`  
`wint_t towupper(wint_t wc);`

機能説明 towupper() 関数は、wint\_t 型の値域を取ります。この範囲は、wchar\_t として表現できる文字で、現在のロケールで有効な文字に対応するワイド文字コードの値であるか、WEOF の値である必要があります。引数が他の値である場合、引数がそのまま返されます。towupper() の引数がワイド文字コードの小文字を表し、対応するワイド文字コードの大文字が存在する (プログラムのロケールカテゴリ LC\_CTYPE にある文字型情報として定義される) 場合には、結果は対応するワイド文字コードの大文字になります。定義域内にあるその他すべての引数はそのまま返されます。

戻り値 正常終了時、towupper() は渡された引数に対応する大文字を返します。そうでない場合、引数をそのまま返します。

エラー 定義されたエラーはありません。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 iswalph(3C), setlocale(3C), tolower(3C), attributes(5)

## ungetwc(3C)

名前 ungetwc – ワイド文字コードを入力ストリームにプッシュバック

形式 #include <stdio.h>  
#include <wchar.h>

```
wint_t ungetwc(wint_t wc, FILE *stream);
```

機能説明 ungetwc() 関数は、*wc* が指定するワイド文字コードに対応する文字を、*stream* が示す入力ストリームにプッシュバック (書き戻し) します。プッシュバックされた文字は、そのストリームに対するその後の読み込み処理で、プッシュ時とは逆の順序で返されます。途中で (*stream* が指すストリームを指定して) ファイルの位置を変える `fseek(3C)`、`fsetpos(3C)`、または `rewind(3C)` 関数を正しく呼び出すと、プッシュバックされた文字はすべて破棄されます。ストリームに対応した外部記憶域の内容は変わりません。

1 文字のプッシュバックは、確実に実行されます。ただし、同じストリームに対し `ungetwc()` を何度も呼び出すと、途中でファイル位置を変える操作を入れないと、プッシュバックが正しく行われなことがあるあります。

*wc* の値が `WEOF` マクロの値に等しいとき、プッシュバックは失敗します。入力ストリームの内容は変わりません。

`ungetwc()` が正しく呼び出されると、ストリーム用の EOF (ファイルの終り) 表示がクリアされます。プッシュバックされた文字がすべて読み込みまたは破棄された後、当該ストリームのファイルポジション表示の値は、プッシュバックが行われる前の値と同じになります。`ungetwc()` が正常に呼び出されるたびに、ファイルポジション表示の値は 1 (またはそれ以上) ずつマイナスされます。呼び出し時点で値が 0 だった場合、呼び出し後の値は予測できません。

戻り値 正常終了時、`ungetwc()` はプッシュバックされた文字に対応するワイド文字コードを返します。正常終了以外の時は `WEOF` を返します。

エラー 以下の場合に `ungetwc()` 関数は異常終了します。

`EILSEQ` 不当な文字シーケンスが検出された、もしくはワイド文字コードが正当な文字に対応していない

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 `read(2)`, `fseek(3C)`, `fsetpos(3C)`, `rewind(3C)`, `setbuf(3C)`, `attributes(5)`

名前 scanf, fscanf, sscanf, vscanf, vscanf, vsscanf – 書式付き入力の変換

形式

```
#include <stdio.h>

int scanf(const char *format, ...);

int fscanf(FILE*stream, const char *format, ...);

int sscanf(const char *s, const char *format, ...);

#include <stdarg.h>
#include <stdio.h>

int vscanf(const char *format, va_list arg);

int vfscanf(FILE *stream, const char *format, va_list arg);

int vsscanf(const char *s, const char *format, va_list arg);
```

機能説明

scanf() 関数は、標準入力ストリーム stdin を読み取ります。

fscanf() 関数は、入力ストリーム *stream* を読み取ります。

sscanf() 関数は、文字列 *s* を読み取ります。

vscanf(), vfscanf(), および vsscanf() 関数はそれぞれ、scanf(), fscanf(), および sscanf() と同等ですが、これらの関数を呼び出すときは、可変数個の引数ではなく、<stdarg.h> ヘッダーで定義されている引数リスト (stdarg(3HEAD) のマニュアルページを参照) を使用します。これらの関数は va\_end() マクロを呼び出しません。そのため、アプリケーションは、これらの関数を使用した後、va\_end(*ap*) を呼び出してクリーンアップ作業を行う必要があります。

これらの関数は、それぞれ、バイトを読み取り、書式にしたがってそれを解釈し、その結果を引数に格納します。これらの関数には、引数として、制御文字列 *format* (以下で説明)、および変換された入力の格納場所を示す *pointer* 群を指定します。書式に対して十分な引数が指定されていない場合、変換の結果は未定義となります。書式が終了しても引数が残っている場合、残りの引数は評価されますが、評価されたとしても無視されます。

変換は、次の未使用の引数にではなく、引数リストの *format* のあとに続く *n* 番目の引数に適用されます。この場合、変換文字 % (後述を参照) は %*n*\$ シーケンスに置換されます。ここで *n* は [1, NL\_ARGMAX] の範囲の 10 進数の整数です。この変換機能は、特定の言語に合わせた順序で引数を選択するように、書式文字列の定義を規定します。書式文字列に変換指定の書式 %*n*\$ が含まれる場合、引数リスト中の番号付けされた引数が、複数回、書式文字列から参照されるかどうかについては不定です。

*format* には、変換指定の形式、つまり % または %*n*\$ のいずれかを含むことができます。ただし、通常は 1 つの *format* 文字列に % と %*n*\$ の両方を指定することはできません。唯一の例外として、%% または %\* を %*n*\$ の形式に入れることができます。

scanf() 関数のすべての形式において、入力文字列中の言語依存の小数点文字を検出できます。小数点文字は、プログラムのロケール (LC\_NUMERIC カテゴリ) によって定義されます。POSIX ロケール および小数点文字が定義されていないロケールでは、小数点文字のデフォルトはピリオド (.) です。

*format* とは、初期シフトの状態が始まるかまたは終わる文字列で、0 個以上の指示語 (もしあれば) で構成されます。各指示語は次のいずれか 1 つで構成されます。

- 1 つまたは複数の空白文字 (スペース、タブ、改行文字、垂直タブ、または用紙送り (form-feed) 文字)
- 通常文字 (% または空白文字は含まない)
- 変換指定

#### 変換指定

各変換指定は % 文字、または %n\$ 文字シーケンスによって導入されます。これらの文字の後には、以下に示す (順番で) 文字列が続きます。

- 代入抑制文字 \* (任意)
- 最大フィールド幅を指定する 0 以外の 10 進数 (任意)
- 受け取るオブジェクトのサイズを表すサイズ修飾子、h、l (小文字のエル)、ll (小文字のエル 2 つ)、または L (任意)。対応する引数が int ではなく short int へのポインタである場合は、変換文字 d、i、および n の前に h を指定する必要があります。また、対応する引数が long int へのポインタである場合は、変換文字 d、i、および n の前に l (小文字のエル) を指定する必要があります。対応する引数が long long int へのポインタである場合は、変換文字 d、i、および n の前に ll (小文字のエル 2 つ) を指定する必要があります。同様に、対応する引数が unsigned int へのポインタではなく unsigned short int へのポインタである場合は、変換文字 o、u、および x の前に h を指定する必要があります。また、対応する引数が unsigned long int へのポインタである場合は、変換文字 o、u、および x の前に l (小文字のエル) を指定する必要があります。対応する引数が unsigned long long int へのポインタである場合は、変換文字 o、u、および x の前に ll (小文字のエル 2 つ) を指定する必要があります。対応する引数が float へのポインタではなく double へのポインタである場合は、変換文字 e、f、および g の前に l (小文字のエル) を指定する必要があります。または、対応する引数が long double へのポインタである場合は、変換文字 e、f、および g の前に L を指定する必要があります。最後に、対応する引数が 文字形式へのポインタではなく wchar\_t へのポインタである場合は、変換文字 c、s、および [ の前に l (小文字のエル) を指定する必要があります。h、l (小文字のエル)、ll (小文字のエル 2 つ)、および L が上記以外の変換文字の前に現われたとき、変換の結果は未定義となります。
- 適用される変換形式を指定する変換文字 (有効な変換文字については、後述の説明を参照してください)。

scanf() 関数は、形式の各指示を順番に実行します。指示にエラーがあった場合、以下に詳しく記述しているように、関数はエラーを返します。エラーは、入力エラー (入力バイトが無効) またはマッチングエラー (入力が不適當) として記述されます。

1 つまたは複数の空白文字で構成される指示は、有効な入力を読み取られなくなるまで、あるいは空白文字でないために読み取れずに残る最初のバイトまで、入力を読み取って実行します。

通常文字の指示は次のように実行されます。次のバイトが入力から読み取られ、指示を含むバイトと比較されます。比較の結果、両者が同じでない場合、指示はエラーで終了し、違いのあった後続のバイトが読み取れずに残ります。

変換指定を表す指示は、一連のマッチング入力シーケンスを定義します (各変換文字については以下に記述)。変換指定は次の手順で実行されます。

変換指定に変換文字 `]`、`c`、`c`、または `n` が含まれる場合を除いて、空白文字の入力 (`isspace(3C)` で指定) はスキップされます。

変換指定に変換文字 `n` が含まれる場合を除いて、項目は入力から読み取られます。入力項目は、入力バイトの最も長いシーケンスとして (変換文字によっては文字またはバイト単位で測定される、指定された最大フィールド幅まで) 定義されます。入力項目幅はマッチングシーケンスの最初のサブシーケンスです。入力項目に続く最初のバイト (もしあれば) は、読み取れずに残ります。入力項目の長さが 0 の場合、変換指定の実行はエラーで終了します。この状態は、ファイルの終わり、コード化エラー、あるいはストリームからの入力を妨げる読み取りエラー (この場合は入力エラー) でない限り、マッチングのエラーです。

変換文字が `%` ではない場合の入力項目 (あるいは変換指定が `%n` の場合の入力バイト数) は、変換文字に適切な形式に変換されます。入力項目がマッチングシーケンスではない場合、変換指定はエラーで終了します。この状態はマッチングエラーです。代入抑制文字が `*` で表された場合を除いて、変換の結果は、以前に変換結果を受け取っていない `format` 引数に続く最初の引数によって、ポイントされたオブジェクトに出力されます (変換指定が `%` によって呼び出されていない場合)。変換指定が文字シーケンス `%n$` によって呼び出された場合は、`n` 番目の引数に出力されます。このオブジェクトが適切な形式ではない場合、または変換の結果が提供された空間に対応できない場合、動作結果は未定義です。

変換文字 以下に、有効な変換文字を示します。

- d 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- i 整数 (符号は任意) と一致します。書式は、`strtoul()` の `base` 引数に値 0 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- o 8 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 8 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。

## vfscanf(3C)

- u 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- x 16 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 16 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- e,f,g 符号付き浮動小数点数に一致します。書式は `strtod(3C)` の変換対象となる文字コードの並びと同じです。サイズ修飾子を指定しない場合、対応する引数は `float` へのポインタである必要があります。
- `printf(3C)` ファミリの関数が、ANSI/IEEE Std 754: 1985 標準を指定できるように、無限および NaN (浮動小数点形式にコード化された 7858 シンボリックエンティティ) 用に文字列表現を生成した場合、`scanf()` ファミリの関数はそれらを入力として認識します。
- s 空白でないバイトシーケンスに一致します。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- l (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。それぞれの文字は、`mbrtowc(3C)` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- [ 複数の文字 (*scanset* という) からなる、空でない文字シーケンスと一致します。この場合、通常行われる先行する空白のスキップは抑制されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL バイトを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- l (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- 変換指定は、*format* 文字列のすべての後続の文字、つまり、対応する右角括弧 (]) まで (または、その括弧も含む) が入ります。括弧内の文字 (*scanlist*) は *scanset* を構成しています。左角括弧の次の文字がサーカンフ

レックス (^) の場合は、サーカンフレックスと右角括弧の間 *scanlist* に入っていないすべての文字が *scanset* を構成しています。変換指定が [ ] または [^] で始まる場合は、この右角括弧は *scanlist* 内に含まれており、次の右角括弧が指定の終了を示す右角括弧になります。*scanlist* に - があり、その - が最初の文字ではない場合、最初の文字が ^ で - が 2 番目の文字ではない場合、あるいは - が最後の文字ではない場合、一致する文字の範囲が表示されます。

c フィールド幅に指定された数 (フィールド幅が変換指定にない場合は 1) の文字シーケンスと一致します。対応する引数は、その文字シーケンスを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。NULL 文字は追加されません。この場合、通常行われる先行する空白のスキップは抑制されます。

l (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよび最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。NULL ワイド文字は追加されません。

p 対応する `printf(3C)` 関数の `%p` 変換によって出力されるシーケンスと同様のシーケンスと一致します。対応する引数は `void` へのポインタである必要があります。入力項目が、同一プログラムの実行中で以前に変換した値である場合は、ポインタは先にその値を変換したときと同様の比較を行います。そうでない場合は、`%p` 変換の動作は未定義です。

n 入力は使用 (変換) されません。対応する引数は、これまでに `scanf()` 関数を呼び出して入力ストリームから読み取ったバイト数を示す整数へのポインタである必要があります。変換指定 `%n` を実行しても、この関数の実行終了時に返される代入数は増加しません。

c 1c と同じです。

s ls と同じです。

% 単一の % と一致します。変換または代入は行われません。完全な変換指定をするには %% を指定してください。

変換指定が正しくない場合の動作は未定義です。

変換文字 E、G、および X はそれぞれ e、g、および x と同じ働きをします。

入力中にファイルの終端になると、変換は終了します。現在の変換指定 (`%n` を除く) に一致するバイト (先行空白を使用している場合はこれを除く) が読み取られる前にファイルが終わると、現在の変換指定の実行は入力エラーで終了します。それ以外の場合は、現在の変換指定の実行がマッチングエラーで終了しない限り、続く変換指定の実行 (もし、あれば) が入力エラーで終了します。

## vfscanf(3C)

	<p>sscanf() の文字列の終端に到達するのは、fscanf() でファイルの終端に到達するのと同様です。</p> <p>入力が衝突して終了した場合、対抗する入力は入力中で読み取られないままになります。変換指定に一致しない場合、後続の空白(改行文字を含む)は読み取られません。リテラル一致および抑制される代入が成功したかどうかは、%n 変換指定によってのみ直接確認できます。</p> <p>fscanf() および scanf() 関数は、stream に関連するファイルの st_atime フィールドを更新用にマークすることができます。st_atime フィールドは、fgetc(3C), fgets(3C), fread(3C), fscanf(), getc(3C), getchar(3C), gets(3C), または scanf() の実行が成功したときに、ungetc(3C) への以前の呼び出しによって与えられていないデータを返す stream を使用して更新用にマークされます。</p>
戻り値	<p>正常終了の場合、これらの関数は一致と代入が成功した入力項目数を返します。一致が実行初期段階で失敗した場合は、0 が返される場合もあります。最初の一致が失敗する前、または最初の変換が行われる前に入力が終わると、EOF が返されます。ストリームが設定されているというエラー表示で読み取りエラーが発生した場合、EOF が返されエラーを表示する errno が設定されます。</p>
エラー	<p>scanf() 関数が失敗、および失敗する可能性がある場合は、fgetc(3C) または fgetwc(3C) を参照してください。</p> <p>さらに、以下の条件で fscanf() は失敗することがあります。</p> <p>EILSEQ            入力バイトシーケンスが有効な文字を形成していない。</p> <p>EINVAL            引数が不足している。</p>
使用法	<p>scanf() 関数を呼び出すアプリケーションに wint_t または wchar_t 型の形式のオブジェクトが存在する場合、これらのオブジェクトを定義する &lt;wchar.h&gt; ヘッダーも含む必要があります。</p>
使用例	<p><b>例 1 使用例 1</b></p> <pre>int i, n; float x; char name[50]; n = scanf("%d%f%s", &amp;i, &amp;x, name)</pre> <p>という呼び出しの場合、入力行は次のようになります。</p> <pre>25 54.32E-1 Hamster</pre> <p>この場合、n に値 3、i に値 25、x に値 5.432 がそれぞれ代入され、name には Hamster の文字列が入ります。</p> <p><b>例 2 使用例 2</b></p> <pre>int i; float x; char name[50]; (void) scanf("%2d%f%d %[0123456789]", &amp;i, &amp;x, name);</pre> <p>という呼び出しの場合、入力行は次のようになります。</p> <pre>56789 0123 56a72</pre>



例 2 使用例 2 (続き)

この場合、*i* に 56、*x* に 789.0 が代入され、0123 はスキップして、*name* には 56\0 が入ります。getchar(3C) への次の呼び出しでは a の文字を返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 fgetc(3C), fgets(3C), fgetwc(3C), fread(3C), isspace(3C), printf(3C), setlocale(3C), stdarg(3HEAD), strtod(3C), strtol(3C), strtoul(3C), wcrntomb(3C), ungetc(3C), attributes(5)

## vscanf(3C)

名前	scanf, fscanf, sscanf, vscanf, vfscanf, vsscanf – 書式付き入力の変換
形式	<pre>#include &lt;stdio.h&gt;  int <b>scanf</b>(const char *<i>format</i>, ...); int <b>fscanf</b>(FILE*<i>stream</i>, const char *<i>format</i>, ...); int <b>sscanf</b>(const char *<i>s</i>, const char *<i>format</i>, ...);  #include &lt;stdarg.h&gt; #include &lt;stdio.h&gt;  int <b>vscanf</b>(const char *<i>format</i>, va_list <i>arg</i>); int <b>vfscanf</b>(FILE *<i>stream</i>, const char *<i>format</i>, va_list <i>arg</i>); int <b>vsscanf</b>(const char *<i>s</i>, const char *<i>format</i>, va_list <i>arg</i>);</pre>
機能説明	<p>scanf() 関数は、標準入力ストリーム stdin を読み取ります。</p> <p>fscanf() 関数は、入力ストリーム <i>stream</i> を読み取ります。</p> <p>sscanf() 関数は、文字列 <i>s</i> を読み取ります。</p> <p>vscanf()、vfscanf()、および vsscanf() 関数はそれぞれ、scanf()、fscanf()、および sscanf() と同等ですが、これらの関数を呼び出すときは、可変数個の引数ではなく、&lt;stdarg.h&gt; ヘッダーで定義されている引数リスト (stdarg(3HEAD) のマニュアルページを参照) を使用します。これらの関数は va_end() マクロを呼び出しません。そのため、アプリケーションは、これらの関数を使用した後、va_end(<i>ap</i>) を呼び出してクリーンアップ作業を行う必要があります。</p> <p>これらの関数は、それぞれ、バイトを読み取り、書式にしたがってそれを解釈し、その結果を引数に格納します。これらの関数には、引数として、制御文字列 <i>format</i> (以下で説明)、および変換された入力の格納場所を示す <i>pointer</i> 群を指定します。書式に対して十分な引数が指定されていない場合、変換の結果は未定義となります。書式が終了しても引数が残っている場合、残りの引数は評価されますが、評価されたとしても無視されます。</p> <p>変換は、次の未使用の引数にではなく、引数リストの <i>format</i> のあとに続く <i>n</i> 番目の引数に適用されます。この場合、変換文字 % (後述を参照) は %<i>n</i>\$ シーケンスに置換されます。ここで <i>n</i> は [1, NL_ARGMAX] の範囲の 10 進数の整数です。この変換機能は、特定の言語に合わせた順序で引数を選択するように、書式文字列の定義を規定します。書式文字列に変換指定の書式 %<i>n</i>\$ が含まれる場合、引数リスト中の番号付けされた引数が、複数回、書式文字列から参照されるかどうかについては不定です。</p> <p><i>format</i> には、変換指定の形式、つまり % または %<i>n</i>\$ のいずれかを含むことができます。ただし、通常は 1 つの <i>format</i> 文字列に % と %<i>n</i>\$ の両方を指定することはできません。唯一の例外として、%% または %* を %<i>n</i>\$ の形式に入れることがあります。</p>

`scanf()` 関数のすべての形式において、入力文字列中の言語依存の小数点文字を検出できます。小数点文字は、プログラムのロケール (LC\_NUMERIC カテゴリ) によって定義されます。POSIX ロケール および小数点文字が定義されていないロケールでは、小数点文字のデフォルトはピリオド (.) です。

*format* とは、初期シフトの状態が始まるかまたは終わる文字列で、0 個以上の指示語 (もしあれば) で構成されます。各指示語は次のいずれか 1 つで構成されます。

- 1 つまたは複数の空白文字 (スペース、タブ、改行文字、垂直タブ、または用紙送り (form-feed) 文字)
- 通常文字 (% または空白文字は含まない)
- 変換指定

#### 変換指定

各変換指定は % 文字、または %n\$ 文字シーケンスによって導入されます。これらの文字の後には、以下に示す (順番で) 文字列が続きます。

- 代入抑制文字 \* (任意)
- 最大フィールド幅を指定する 0 以外の 10 進数 (任意)
- 受け取るオブジェクトのサイズを表すサイズ修飾子、h、l (小文字のエル)、ll (小文字のエル 2 つ)、または L (任意)。対応する引数が `int` ではなく `short int` へのポインタである場合は、変換文字 d、i、および n の前に h を指定する必要があります。また、対応する引数が `long int` へのポインタである場合は、変換文字 d、i、および n の前に l (小文字のエル) を指定する必要があります。対応する引数が `long long int` へのポインタである場合は、変換文字 d、i、および n の前に ll (小文字のエル 2 つ) を指定する必要があります。同様に、対応する引数が `unsigned int` へのポインタではなく `unsigned short int` へのポインタである場合は、変換文字 o、u、および x の前に h を指定する必要があります。また、対応する引数が `unsigned long int` へのポインタである場合は、変換文字 o、u、および x の前に l (小文字のエル) を指定する必要があります。対応する引数が `unsigned long long int` へのポインタである場合は、変換文字 o、u、および x の前に ll (小文字のエル 2 つ) を指定する必要があります。対応する引数が `float` へのポインタではなく `double` へのポインタである場合は、変換文字 e、f、および g の前に l (小文字のエル) を指定する必要があります。または、対応する引数が `long double` へのポインタである場合は、変換文字 e、f、および g の前に L を指定する必要があります。最後に、対応する引数が文字形式へのポインタではなく `wchar_t` へのポインタである場合は、変換文字 c、s、および [ の前に l (小文字のエル) を指定する必要があります。h、l (小文字のエル)、ll (小文字のエル 2 つ)、および L が上記以外の変換文字の前に現われたとき、変換の結果は未定義となります。
- 適用される変換形式を指定する変換文字 (有効な変換文字については、後述の説明を参照してください)。

`scanf()` 関数は、形式の各指示を順番に実行します。指示にエラーがあった場合、以下に詳しく記述しているように、関数はエラーを返します。エラーは、入力エラー (入力バイトが無効) またはマッチングエラー (入力ที่ไม่適當) として記述されます。

## vscanf(3C)

1 つまたは複数の空白文字で構成される指示は、有効な入力を読み取られなくなるまで、あるいは空白文字でないために読み取れずに残る最初のバイトまで、入力を読み取って実行します。

通常文字の指示は次のように実行されます。次のバイトが入力から読み取られ、指示を含むバイトと比較されます。比較の結果、両者が同じでない場合、指示はエラーで終了し、違いのあった後続のバイトが読み取れずに残ります。

変換指定を表す指示は、一連のマッチング入力シーケンスを定義します (各変換文字については以下に記述)。変換指定は次の手順で実行されます。

変換指定に変換文字 `l`、`c`、`c`、または `n` が含まれる場合を除いて、空白文字の入力 (`isspace(3C)` で指定) はスキップされます。

変換指定に変換文字 `n` が含まれる場合を除いて、項目は入力から読み取られます。入力項目は、入力バイトの最も長いシーケンスとして (変換文字によっては文字またはバイト単位で測定される、指定された最大フィールド幅まで) 定義されます。入力項目幅はマッチングシーケンスの最初のサブシーケンスです。入力項目に続く最初のバイト (もしあれば) は、読み取れずに残ります。入力項目の長さが 0 の場合、変換指定の実行はエラーで終了します。この状態は、ファイルの終わり、コード化エラー、あるいはストリームからの入力を妨げる読み取りエラー (この場合は入力エラー) でない限り、マッチングのエラーです。

変換文字が `%` ではない場合の入力項目 (あるいは変換指定が `%n` の場合の入力バイト数) は、変換文字に適切な形式に変換されます。入力項目がマッチングシーケンスではない場合、変換指定はエラーで終了します。この状態はマッチングエラーです。代入抑制文字が `*` で表された場合を除いて、変換の結果は、以前に変換結果を受け取っていない `format` 引数に続く最初の引数によって、ポイントされたオブジェクトに出力されます (変換指定が `%` によって呼び出されていない場合)。変換指定が文字シーケンス `%n$` によって呼び出された場合は、`n` 番目の引数に出力されます。このオブジェクトが適切な形式ではない場合、または変換の結果が提供された空間に対応できない場合、動作結果は未定義です。

変換文字 以下に、有効な変換文字を示します。

- d 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- i 整数 (符号は任意) と一致します。書式は、`strtoul()` の `base` 引数に値 0 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- o 8 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 8 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。

- u 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- x 16 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 16 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- e,f,g 符号付き浮動小数点数に一致します。書式は `strtod(3C)` の変換対象となる文字コードの並びと同じです。サイズ修飾子を指定しない場合、対応する引数は `float` へのポインタである必要があります。
- `printf(3C)` ファミリの関数が、ANSI/IEEE Std 754: 1985 標準を指定できるように、無限および NaN (浮動小数点形式にコード化された 7858 シンボリックエンティティ) 用に文字列表現を生成した場合、`scanf()` ファミリの関数はそれらを入力として認識します。
- s 空白でないバイトシーケンスに一致します。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- 1 (小文字のエル) 修飾子が与えられた場合、入力 は初期シフト状態で始まる文字シーケンスです。それぞれの文字は、`mbrtowc(3C)` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- [ 複数の文字 (*scanset* という) からなる、空でない文字シーケンスと一致します。この場合、通常行われる先行する空白のスキップは抑制されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL バイトを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- 1 (小文字のエル) 修飾子が与えられた場合、入力 は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- 変換指定は、*format* 文字列のすべての後続の文字、つまり、対応する右角括弧 (]) まで (または、その括弧も含む) が入ります。括弧内の文字 (*scanlist*) は *scanset* を構成しています。左角括弧の次の文字がサーカンフ

レックス (^) の場合は、サーカンフレックスと直角括弧の間 *scanlist* に入っていないすべての文字が *scanset* を構成しています。変換指定が [ ] または [^] で始まる場合は、この直角括弧は *scanlist* 内に含まれており、次の直角括弧が指定の終了を示す直角括弧になります。*scanlist* に - があり、その - が最初の文字ではない場合、最初の文字が ^ で - が 2 番目の文字ではない場合、あるいは - が最後の文字ではない場合、一致する文字の範囲が表示されます。

**c** フィールド幅に指定された数 (フィールド幅が変換指定にない場合は 1) の文字シーケンスと一致します。対応する引数は、その文字シーケンスを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。NULL 文字は追加されません。この場合、通常行われる先行する空白のスキップは抑制されます。

l (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよび最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。NULL ワイド文字は追加されません。

**p** 対応する `printf(3C)` 関数の `%p` 変換によって出力されるシーケンスと同様のシーケンスと一致します。対応する引数は `void` へのポインタである必要があります。入力項目が、同一プログラムの実行中で以前に変換した値である場合は、ポインタは先にその値を変換したときと同様の比較を行います。そうでない場合は、`%p` 変換の動作は未定義です。

**n** 入力は使用 (変換) されません。対応する引数は、これまでに `scanf()` 関数を呼び出して入力ストリームから読み取ったバイト数を示す整数へのポインタである必要があります。変換指定 `%n` を実行しても、この関数の実行終了時に返される代入数は増加しません。

**C** `lc` と同じです。

**S** `ls` と同じです。

**%** 単一の `%` と一致します。変換または代入は行われません。完全な変換指定をするには `%%` を指定してください。

変換指定が正しくない場合の動作は未定義です。

変換文字 `E`, `G`, および `X` はそれぞれ `e`, `g`, および `x` と同じ働きをします。

入力中にファイルの終端になると、変換は終了します。現在の変換指定 (`%n` を除く) に一致するバイト (先行空白を使用している場合はこれを除く) が読み取られる前にファイルが終わると、現在の変換指定の実行は入力エラーで終了します。それ以外の場合は、現在の変換指定の実行がマッチングエラーで終了しない限り、続く変換指定の実行 (もし、あれば) が入力エラーで終了します。

`sscanf()` の文字列の終端に到達するのは、`fscanf()` でファイルの終端に到達するのと同様です。

入力が衝突して終了した場合、対抗する入力は入力中で読み取られないままになります。変換指定に一致しない場合、後続の空白 (改行文字を含む) は読み取られません。リテラル一致および抑制される代入が成功したかどうかは、`%n` 変換指定によってのみ直接確認できます。

`fscanf()` および `scanf()` 関数は、*stream* に関連するファイルの `st_atime` フィールドを更新用にマークすることができます。 `st_atime` フィールドは、`fgetc(3C)`, `fgets(3C)`, `fread(3C)`, `fscanf()`, `getc(3C)`, `getchar(3C)`, `gets(3C)`, または `scanf()` の実行が成功したときに、`ungetc(3C)` への以前の呼び出しによって与えられていないデータを返す *stream* を使用して更新用にマークされます。

**戻り値** 正常終了の場合、これらの関数は一致と代入が成功した入力項目数を返します。一致が実行初期段階で失敗した場合は、0 が返される場合もあります。最初の一致が失敗する前、または最初の変換が行われる前に入力が終わると、EOF が返されます。ストリームが設定されているというエラー表示で読み取りエラーが発生した場合、EOF が返されエラーを表示する `errno` が設定されます。

**エラー** `scanf()` 関数が失敗、および失敗する可能性がある場合は、`fgetc(3C)` または `fgetwc(3C)` を参照してください。

さらに、以下の条件で `fscanf()` は失敗することがあります。

`EILSEQ` 入力バイトシーケンスが有効な文字を形成していない。

`EINVAL` 引数が不足している。

**使用法** `scanf()` 関数を呼び出すアプリケーションに `wint_t` または `wchar_t` 型の形式のオブジェクトが存在する場合、これらのオブジェクトを定義する `<wchar.h>` ヘッダーも含む必要があります。

**使用例** 例 1 使用例 1

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name)
```

という呼び出しの場合、入力行は次のようになります。

```
25 54.32E-1 Hamster
```

この場合、*n* に値 3、*i* に値 25、*x* に値 5.432 がそれぞれ代入され、*name* には Hamster の文字列が入ります。

例 2 使用例 2

```
int i; float x; char name[50];
(void) scanf("%2d%f*d %[0123456789]", &i, &x, name);
```

という呼び出しの場合、入力行は次のようになります。

```
56789 0123 56a72
```

## vscanf(3C)

### 例 2 使用例 2 (続き)

この場合、*i* に 56、*x* に 789.0 が代入され、0123 はスキップして、*name* には 56\0 が入ります。getchar(3C) への次の呼び出しでは a の文字を返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 fgetc(3C), fgets(3C), fgetwc(3C), fread(3C), isspace(3C), printf(3C), setlocale(3C), stdarg(3HEAD), strtod(3C), strtol(3C), strtoul(3C), wcrctomb(3C), ungetc(3C), attributes(5)



名前 scanf, fscanf, sscanf, vscanf, vfscanf, vsscanf – 書式付き入力の変換

形式

```
#include <stdio.h>

int scanf(const char *format, ...);

int fscanf(FILE*stream, const char *format, ...);

int sscanf(const char *s, const char *format, ...);

#include <stdarg.h>
#include <stdio.h>

int vscanf(const char *format, va_list arg);

int vfscanf(FILE *stream, const char *format, va_list arg);

int vsscanf(const char *s, const char *format, va_list arg);
```

機能説明

scanf() 関数は、標準入力ストリーム stdin を読み取ります。

fscanf() 関数は、入力ストリーム *stream* を読み取ります。

sscanf() 関数は、文字列 *s* を読み取ります。

vscanf(), vfscanf(), および vsscanf() 関数はそれぞれ、scanf(), fscanf(), および sscanf() と同等ですが、これらの関数を呼び出すときは、可変数個の引数ではなく、<stdarg.h> ヘッダーで定義されている引数リスト (stdarg(3HEAD) のマニュアルページを参照) を使用します。これらの関数は va\_end() マクロを呼び出しません。そのため、アプリケーションは、これらの関数を使用した後、va\_end(*ap*) を呼び出してクリーンアップ作業を行う必要があります。

これらの関数は、それぞれ、バイトを読み取り、書式にしたがってそれを解釈し、その結果を引数に格納します。これらの関数には、引数として、制御文字列 *format* (以下で説明)、および変換された入力の格納場所を示す *pointer* 群を指定します。書式に対して十分な引数が指定されていない場合、変換の結果は未定義となります。書式が終了しても引数が残っている場合、残りの引数は評価されますが、評価されたとしても無視されます。

変換は、次の未使用の引数にではなく、引数リストの *format* のあとに続く *n* 番目の引数に適用されます。この場合、変換文字 % (後述を参照) は %*n*\$ シーケンスに置換されます。ここで *n* は [1, NL\_ARGMAX] の範囲の 10 進数の整数です。この変換機能は、特定の言語に合わせた順序で引数を選択するように、書式文字列の定義を規定します。書式文字列に変換指定の書式 %*n*\$ が含まれる場合、引数リスト中の番号付けされた引数が、複数回、書式文字列から参照されるかどうかについては不定です。

*format* には、変換指定の形式、つまり % または %*n*\$ のいずれかを含むことができます。ただし、通常は 1 つの *format* 文字列に % と %*n*\$ の両方を指定することはできません。唯一の例外として、%% または %\* を %*n*\$ の形式に入れることがあります。

scanf() 関数のすべての形式において、入力文字列中の言語依存の小数点文字を検出できます。小数点文字は、プログラムのロケール (LC\_NUMERIC カテゴリ) によって定義されます。POSIX ロケール および小数点文字が定義されていないロケールでは、小数点文字のデフォルトはピリオド (.) です。

*format* とは、初期シフトの状態が始まるかまたは終わる文字列で、0 個以上の指示語 (もしあれば) で構成されます。各指示語は次のいずれか 1 つで構成されます。

- 1 つまたは複数の空白文字 (スペース、タブ、改行文字、垂直タブ、または用紙送り (form-feed) 文字)
- 通常文字 (% または空白文字は含まない)
- 変換指定

#### 変換指定

各変換指定は % 文字、または %n\$ 文字シーケンスによって導入されます。これらの文字の後には、以下に示す (順番で) 文字列が続きます。

- 代入抑制文字 \* (任意)
- 最大フィールド幅を指定する 0 以外の 10 進数 (任意)
- 受け取るオブジェクトのサイズを表すサイズ修飾子、h、l (小文字のエル)、ll (小文字のエル 2 つ)、または L (任意)。対応する引数が int ではなく short int へのポインタである場合は、変換文字 d、i、および n の前に h を指定する必要があります。また、対応する引数が long int へのポインタである場合は、変換文字 d、i、および n の前に l (小文字のエル) を指定する必要があります。対応する引数が long long int へのポインタである場合は、変換文字 d、i、および n の前に ll (小文字のエル 2 つ) を指定する必要があります。同様に、対応する引数が unsigned int へのポインタではなく unsigned short int へのポインタである場合は、変換文字 o、u、および x の前に h を指定する必要があります。また、対応する引数が unsigned long int へのポインタである場合は、変換文字 o、u、および x の前に l (小文字のエル) を指定する必要があります。対応する引数が unsigned long long int へのポインタである場合は、変換文字 o、u、および x の前に ll (小文字のエル 2 つ) を指定する必要があります。対応する引数が float へのポインタではなく double へのポインタである場合は、変換文字 e、f、および g の前に l (小文字のエル) を指定する必要があります。または、対応する引数が long double へのポインタである場合は、変換文字 e、f、および g の前に L を指定する必要があります。最後に、対応する引数が 文字形式へのポインタではなく wchar\_t へのポインタである場合は、変換文字 c、s、および [ の前に l (小文字のエル) を指定する必要があります。h、l (小文字のエル)、ll (小文字のエル 2 つ)、および L が上記以外の変換文字の前に現われたとき、変換の結果は未定義となります。
- 適用される変換形式を指定する変換文字 (有効な変換文字については、後述の説明を参照してください)。

scanf() 関数は、形式の各指示を順番に実行します。指示にエラーがあった場合、以下に詳しく記述しているように、関数はエラーを返します。エラーは、入力エラー (入力バイトが無効) またはマッチングエラー (入力が不適當) として記述されます。

1 つまたは複数の空白文字で構成される指示は、有効な入力を読み取られなくなるまで、あるいは空白文字でないために読み取れずに残る最初のバイトまで、入力を読み取って実行します。

通常文字の指示は次のように実行されます。次のバイトが入力から読み取られ、指示を含むバイトと比較されます。比較の結果、両者が同じでない場合、指示はエラーで終了し、違いのあった後続のバイトが読み取れずに残ります。

変換指定を表す指示は、一連のマッチング入力シーケンスを定義します (各変換文字については以下に記述)。変換指定は次の手順で実行されます。

変換指定に変換文字 `]`、`c`、`c`、または `n` が含まれる場合を除いて、空白文字の入力 (`isspace(3C)` で指定) はスキップされます。

変換指定に変換文字 `n` が含まれる場合を除いて、項目は入力から読み取られます。入力項目は、入力バイトの最も長いシーケンスとして (変換文字によっては文字またはバイト単位で測定される、指定された最大フィールド幅まで) 定義されます。入力項目幅はマッチングシーケンスの最初のサブシーケンスです。入力項目に続く最初のバイト (もしあれば) は、読み取れずに残ります。入力項目の長さが 0 の場合、変換指定の実行はエラーで終了します。この状態は、ファイルの終わり、コード化エラー、あるいはストリームからの入力を妨げる読み取りエラー (この場合は入力エラー) でない限り、マッチングのエラーです。

変換文字が `%` ではない場合の入力項目 (あるいは変換指定が `%n` の場合の入力バイト数) は、変換文字に適切な形式に変換されます。入力項目がマッチングシーケンスではない場合、変換指定はエラーで終了します。この状態はマッチングエラーです。代入抑制文字が `*` で表された場合を除いて、変換の結果は、以前に変換結果を受け取っていない `format` 引数に続く最初の引数によって、ポイントされたオブジェクトに出力されます (変換指定が `%` によって呼び出されていない場合)。変換指定が文字シーケンス `%n$` によって呼び出された場合は、`n` 番目の引数に出力されます。このオブジェクトが適切な形式ではない場合、または変換の結果が提供された空間に対応できない場合、動作結果は未定義です。

変換文字 以下に、有効な変換文字を示します。

- d 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- i 整数 (符号は任意) と一致します。書式は、`strtoul()` の `base` 引数に値 0 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`int` へのポインタである必要があります。
- o 8 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の `base` 引数に値 8 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。

## vsscanf(3C)

- u 10 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 10 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- x 16 進数の整数 (符号は任意) と一致します。書式は、`strtoul(3C)` の *base* 引数に値 16 を指定したときの変換対象となる文字コードの並びと同じものである必要があります。サイズ修飾子を指定しない場合、対応する引数は、`unsigned int` へのポインタである必要があります。
- e,f,g 符号付き浮動小数点数に一致します。書式は `strtod(3C)` の変換対象となる文字コードの並びと同じです。サイズ修飾子を指定しない場合、対応する引数は `float` へのポインタである必要があります。
- `printf(3C)` ファミリの関数が、ANSI/IEEE Std 754: 1985 標準を指定できるように、無限および NaN (浮動小数点形式にコード化された 7858 シンボリックエンティティ) 用に文字列表現を生成した場合、`scanf()` ファミリの関数はそれらを入力として認識します。
- s 空白でないバイトシーケンスに一致します。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- l (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。それぞれの文字は、`mbrtowc(3C)` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- [ 複数の文字 (*scanset* という) からなる、空でない文字シーケンスと一致します。この場合、通常行われる先行する空白のスキップは抑制されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL バイトを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。
- l (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよびその最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。
- 変換指定は、*format* 文字列のすべての後続の文字、つまり、対応する右角括弧 (]) まで (または、その括弧も含む) が入ります。括弧内の文字 (*scanlist*) は *scanset* を構成しています。左角括弧の次の文字がサーカンフ

レックス (^) の場合は、サーカンフレックスと右角括弧の間 *scanlist* に入っていないすべての文字が *scanset* を構成しています。変換指定が [ ] または [^] で始まる場合は、この右角括弧は *scanlist* 内に含まれており、次の右角括弧が指定の終了を示す右角括弧になります。*scanlist* に - があり、その - が最初の文字ではない場合、最初の文字が ^ で - が 2 番目の文字ではない場合、あるいは - が最後の文字ではない場合、一致する文字の範囲が表示されます。

**c** フィールド幅に指定された数 (フィールド幅が変換指定にない場合は 1) の文字シーケンスと一致します。対応する引数は、その文字シーケンスを十分格納できるだけの大きさの `char`, `signed char`, または `unsigned char` の並びの最初のバイトへのポインタである必要があります。NULL 文字は追加されません。この場合、通常行われる先行する空白のスキップは抑制されます。

1 (小文字のエル) 修飾子が与えられた場合、入力は初期シフト状態で始まる文字シーケンスです。その並びにあるそれぞれの文字は、`mbrtowc()` 関数への呼び出しと同じようにワイド文字に変換され、変換状態は、最初の文字が変換される前に 0 に初期化された `mbstate_t` オブジェクトによって記述されます。対応する引数は、その文字シーケンスおよび最後に自動的に追加される NULL 文字コードを十分格納できるだけの大きさの `wchar_t` の配列へのポインタである必要があります。NULL ワイド文字は追加されません。

**p** 対応する `printf(3C)` 関数の `%p` 変換によって出力されるシーケンスと同様のシーケンスと一致します。対応する引数は `void` へのポインタである必要があります。入力項目が、同一プログラムの実行中で以前に変換した値である場合は、ポインタは先にその値を変換したときと同様の比較を行います。そうでない場合は、`%p` 変換の動作は未定義です。

**n** 入力は使用 (変換) されません。対応する引数は、これまでに `scanf()` 関数を呼び出して入力ストリームから読み取ったバイト数を示す整数へのポインタである必要があります。変換指定 `%n` を実行しても、この関数の実行終了時に返される代入数は増加しません。

**c** 1c と同じです。

**s** 1s と同じです。

**%** 単一の % と一致します。変換または代入は行われません。完全な変換指定をするには %% を指定してください。

変換指定が正しくない場合の動作は未定義です。

変換文字 E、G、および X はそれぞれ e、g、および x と同じ働きをします。

入力中にファイルの終端になると、変換は終了します。現在の変換指定 (`%n` を除く) に一致するバイト (先行空白を使用している場合はこれを除く) が読み取られる前にファイルが終わると、現在の変換指定の実行は入力エラーで終了します。それ以外の場合は、現在の変換指定の実行がマッチングエラーで終了しない限り、続く変換指定の実行 (もし、あれば) が入力エラーで終了します。

## vsscanf(3C)

	<p><code>vsscanf()</code> の文字列の終端に到達するのは、<code>fscanf()</code> でファイルの終端に到達するのと同様です。</p> <p>入力が衝突して終了した場合、対抗する入力は入力中で読み取られないままになります。変換指定に一致しない場合、後続の空白(改行文字を含む)は読み取られません。リテラル一致および抑制される代入が成功したかどうかは、<code>%n</code> 変換指定によってのみ直接確認できます。</p> <p><code>fscanf()</code> および <code>scanf()</code> 関数は、<i>stream</i> に関連するファイルの <code>st_atime</code> フィールドを更新用にマークすることができます。 <code>st_atime</code> フィールドは、<code>fgetc(3C)</code>, <code>fgets(3C)</code>, <code>fread(3C)</code>, <code>fscanf()</code>, <code>getc(3C)</code>, <code>getchar(3C)</code>, <code>gets(3C)</code>, または <code>scanf()</code> の実行が成功したときに、<code>ungetc(3C)</code> への以前の呼び出しによって与えられていないデータを返す <i>stream</i> を使用して更新用にマークされます。</p>
戻り値	<p>正常終了の場合、これらの関数は一致と代入が成功した入力項目数を返します。一致が実行初期段階で失敗した場合は、0 が返される場合もあります。最初の一致が失敗する前、または最初の変換が行われる前に入力が終わると、EOF が返されます。ストリームが設定されているというエラー表示で読み取りエラーが発生した場合、EOF が返されエラーを表示する <code>errno</code> が設定されます。</p>
エラー	<p><code>scanf()</code> 関数が失敗、および失敗する可能性がある場合は、<code>fgetc(3C)</code> または <code>fgetwc(3C)</code> を参照してください。</p> <p>さらに、以下の条件で <code>fscanf()</code> は失敗することがあります。</p> <p>EILSEQ            入力バイトシーケンスが有効な文字を形成していない。</p> <p>EINVAL            引数が不足している。</p>
使用法	<p><code>scanf()</code> 関数を呼び出すアプリケーションに <code>wint_t</code> または <code>wchar_t</code> 型の形式のオブジェクトが存在する場合、これらのオブジェクトを定義する <code>&lt;wchar.h&gt;</code> ヘッダーも含む必要があります。</p>
使用例	<p><b>例 1 使用例 1</b></p> <pre>int i, n; float x; char name[50]; n = scanf("%d%f%s", &amp;i, &amp;x, name)</pre> <p>という呼び出しの場合、入力行は次のようになります。</p> <pre>25 54.32E-1 Hamster</pre> <p>この場合、<i>n</i> に値 3、<i>i</i> に値 25、<i>x</i> に値 5.432 がそれぞれ代入され、<i>name</i> には Hamster の文字列が入ります。</p> <p><b>例 2 使用例 2</b></p> <pre>int i; float x; char name[50]; (void) scanf("%2d%f*d %[0123456789]", &amp;i, &amp;x, name);</pre> <p>という呼び出しの場合、入力行は次のようになります。</p> <pre>56789 0123 56a72</pre>

## 例 2 使用例 2 (続き)

この場合、*i* に 56、*x* に 789.0 が代入され、0123 はスキップして、*name* には 56\0 が入ります。getchar(3C) への次の呼び出しでは a の文字を返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 fgetc(3C), fgets(3C), fgetwc(3C), fread(3C), isspace(3C), printf(3C), setlocale(3C), stdarg(3HEAD), strtod(3C), strtol(3C), strtoul(3C), wcrntomb(3C), ungetc(3C), attributes(5)

## watof(3C)

名前	wcstod, wstod, watof – ワイド文字列を倍精度浮動小数点数に変換
形式	<pre>#include &lt;wchar.h&gt;  double wcstod(const wchar_t *nptr, wchar_t **endptr); double wstod(const wchar_t *nptr, wchar_t **endptr); double watof(wchar_t *nptr);</pre>
機能説明	<p>wcstod() と wstod() の両関数は、<i>nptr</i> が示すワイド文字列の先頭部分を double 型の表現に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。初めの部分は、iswspace(3C) で識別される空白のワイド文字コードの並びで、空の場合もあります。次に変換対象となる文字コードの並びで、浮動小数点数として解釈されます。最後の部分は、解釈不能なワイド文字コードの並びで、これには入力文字列の終わりの NULL ワイド文字コードも含まれます。分割後、変換対象となる文字コードの並びを浮動小数点数に変換しようと試み、その結果を返します。</p> <p>変換対象となる文字コードの並びとして期待される形式は、先頭が + または - の符号 (なくても可) で、その後が小数点を含むまたは含まない 1 つ以上の数字の集まりで、必要であれば最後に指数部が続きます。指数部は、文字 e または E で、その後にはオプションとして符号やいくつかの 10 進数を付加することもできます。変換対象となる文字コードの並びとは、入力ワイド文字列中の最初の空白でないワイド文字コードで始まる、期待された形式の、最長のワイド文字コードの並びであると定義できます。入力ワイド文字列が空の場合、または空白を表すワイド文字コードだけからなる場合、または最初の空白でないワイド文字コードが符号でも数字でも小数点でもない場合、変換対象となる文字コードの並びにはワイド文字コードが 1 つも含まれません。</p> <p>変換対象となる文字コードの並びの形式が期待どおりの場合、最初の数字または小数点 (どちらか最初に現れた方) で始まる一連のワイド文字コードは、C 言語で定義される浮動小数点数であると解釈されます。C 言語の定義と異なる点は、ピリオドの代わりに小数点が使われること、指数も小数点も現れなければワイド文字列中の最後の数字の後に小数点が続くと見なされることです。変換対象となる文字コードの並びが負の符号 (-) で始まっている場合、変換の結果として得られた値は負となります。<i>endptr</i> が示すオブジェクトには、最終部分のワイド文字列へのポインタが書き込まれます。ただし <i>endptr</i> が NULL ポインタの場合を除きます。</p> <p>小数点はプログラムのロケール (LC_NUMERIC カテゴリ) 中に定義されます。POSIX ロケール、および小数点が定義されていないロケールにおいては、小数点のデフォルト値はピリオド (.) となります。</p> <p>ロケールが POSIX 以外の場合、そのシステム固有の他の変換対象となる文字コードの並び形式も使用できます。</p> <p>変換対象となる文字コードの並びが空の場合、または期待された形式ではない場合、変換処理は行われません。またこのとき、<i>endptr</i> が示すオブジェクトには、<i>nptr</i> の値が書き込まれます。ただし <i>endptr</i> が NULL ポインタの場合を除きます。</p> <p>watof(str) 関数は wstod(str, (wchar_t **)NULL) と同じです。</p>



戻り値 `wcstod()` および `wstod()` 関数は、変換された値 (もしあれば) を返します。変換が行われなかった場合、0 が返されます。このとき、`errno` は `EINVAL` の値に設定されることがあります。

正しい値が表現可能な値の範囲外であれば、`±HUGE_VAL` のどれか (値の符号による) が返され、`errno` は `ERANGE` の値に設定されます。

正しい値がアンダフローを起こしてしまうのであれば、0 が返され、`errno` は `ERANGE` の値に設定されます。

エラー `wcstod()` および `wstod()` 関数は、以下の状態のときエラーを返します。

`ERANGE` 返されるべき値が、オーバフローまたはアンダフローを起こしてしまう

`wcstod()` および `wcstod()` 関数は、以下の状態のときエラーを返します。

`EINVAL` 変換が実行できない

使用法 エラー発生時には 0 が返されますが、これは正常終了の場合にも返される値です。したがってエラー発生の有無を知りたいアプリケーションは、`errno` を 0 に設定して `wcstod()` または `wstod()` を呼び出し、処理実行後に `errno` の値を調べる必要があります。値が 0 以外であれば、エラーが起こったものと判断できます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 `iswspace(3C)`, `localeconv(3C)`, `scanf(3C)`, `setlocale(3C)`, `wcstol(3C)`, `attributes(5)`

## watoi(3C)

名前	wcstol, wstol, watol, watoll, watoi – ワイド文字列を整数に変換
形式	<pre>#include &lt;wchar.h&gt;  long int <b>wcstol</b>(const wchar_t *nptr, wchar_t **endptr, int base);  #include &lt;widec.h&gt;  long int <b>wstol</b>(const wchar_t *nptr, wchar_t **endptr, int base);  long <b>watol</b>(wchar_t *nptr);  long long <b>watoll</b>(wchar_t *nptr);  int <b>watoi</b>(wchar_t *nptr);</pre>
機能説明	<p><code>wcstol()</code> と <code>wstol()</code> の両関数は、<code>nptr</code> が示すワイド文字列の先頭部分を <code>long int</code> 型の整数表現に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。初めの部分は、<code>iswspace(3C)</code> で識別される空白のワイド文字コードの並びで、空の場合もあります。次が変換対象となる文字コードの並びで、<code>base</code> の値から決定される指数表現の整数として解釈されます。最後の部分は、解釈不能なワイド文字コードの並びで、これには入力文字列の終わりの NULL ワイド文字コードも含まれます。分割後、変換対象となる文字コードの並びを整数に変換しようと試み、その結果を返します。</p> <p><code>base</code> の値が 0 のとき、変換対象となる文字コードの並びの値は 10 進、8 進、または 16 進の定数であるものと期待されます。定数には + または - の符号が付いていてもかまいません。10 進定数は 0 以外の数字で始まり、いくつかの 10 進数で構成されます。8 進定数は接頭辞 0 で認識され、その後には 0 から 7 までの数字をいくつか続けることができます。16 進定数は、接頭辞 0x または 0X で認識され、その後には 0 から 9 までの数字および a (または A) から f (または F) までの文字がいくつか続きます。</p> <p><code>base</code> の値が 2 から 36 までの範囲のとき、変換対象となる文字コードの並びの形式は <code>base</code> が示す基数を用いた整数を表す一連の数字または文字であると期待されます。先頭に + または - の符号が付加されることはありませんが、整数の接尾辞は含まれません。文字 a (または A) から z (または Z) は、それぞれ 10 から 35 までの値に対応します。<code>base</code> の値以内の範囲の文字だけが、数値に対応することができます。<code>base</code> の値が 16 のとき、値を表す一連の数字や文字の前に、0x または 0X のワイド文字コード表現を付加することができます。符号が存在していれば、その後となります。</p> <p>変換対象となる文字コードの並びとは、入力ワイド文字列中の最初の空白以外のワイド文字コードで始まる、期待された形式の、最長のワイド文字コードの並びであると定義できます。入力ワイド文字列が空の場合、または空白を表すワイド文字コードだけからなる場合、または最初の空白以外のワイド文字コードが符号でもなく指定可能な数字や文字でもない場合、変換対象となる文字コードの並びにはワイド文字コードが 1 つも含まれません。</p> <p>変換対象となる文字コードの並びの形式が期待どおりのものであり、<code>base</code> の値が 0 の場合、最初の数字で始まる一連のワイド文字コードは、整数であると解釈されます。変換対象となる文字コードの並びの形式が期待どおりのものであり、<code>base</code> の値が 2 から 36 までの範囲にある場合、その値を基数として変換が行われます。文字に対しては、前述したような方法で数値が割り当てられます。変換対象となる文字コード</p>

の並びが負の符号 (-) で始まっている場合、変換の結果として得られる値は負となります。 *endptr* が示すオブジェクトには、最終部分のワイド文字列へのポインタが書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

ロケールが POSIX 以外の場合、そのシステム固有の他の変換対象となる文字コードの並び形式も使用できます。

変換対象となる文字コードの並びが空の場合、または期待された形式ではない場合、変換処理は行われません。またこのとき、 *endptr* が示すオブジェクトには、 *nptr* の値が書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

*watol*( ) 関数は *wstol*(*str*, (*wchar\_t* \*\*)NULL, 10) と同機能です。

*watoll*( ) 関数は longlong 型の *watol*( ) です。

*watoi*( ) 関数は (int)*watol*( ) と同機能です。

戻り値 処理が正常に終了すると、 *wcstol*( ) および *wstol*( ) は、変換された値 (もしあれば) を返します。変換が行われなかった場合、0 が返されます。このとき、 *errno* はエラー状態を示す値に設定されることがあります。正しい値が表現可能な値の範囲外であれば、 {LONG\_MAX} または {LONG\_MIN} のどちらか (値の符号による) が返され、 *errno* は ERANGE の値に設定されます。

エラー *wcstol*( ) および *wstol*( ) 関数は、以下の条件のとき異常終了します。

EINVAL *base* の値はサポートされていない

ERANGE 返されるべき値は表現不能である

また *wcstol*( ) および *wstol*( ) 関数は、以下の状態のときエラーを返します。

EINVAL 変換が実行できない

属性 次の属性については *attributes*(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 *iswalph*(3C), *iswspace*(3C), *scanf*(3C), *wcstod*(3C), *attributes*(5)

注意事項 エラー発生時には 0、 {LONG\_MIN}、 および {LONG\_MAX} が返されますが、これらは正常終了の場合にも返される値です。したがってエラー発生の有無を知りたいアプリケーションは、 *errno* を 0 に設定して *wcstol*( ) または *wstol*( ) を呼び出し、処理実行後に *errno* の値を調べる必要があります。値が 0 以外であれば、エラーが起こったものと判断できます。

代入時に必要であれば、または明示的に指定されていれば、 longlong 型から long 型への切り捨てが発生することがあります。

## watol(3C)

名前	wcstol, wstol, watol, watoll, watoi – ワイド文字列を整数に変換
形式	<pre>#include &lt;wchar.h&gt;  long int <b>wcstol</b>(const wchar_t *nptr, wchar_t **endptr, int base);  #include &lt;widec.h&gt;  long int <b>wstol</b>(const wchar_t *nptr, wchar_t **endptr, int base);  long <b>watol</b>(wchar_t *nptr);  long long <b>watoll</b>(wchar_t *nptr);  int <b>watoi</b>(wchar_t *nptr);</pre>
機能説明	<p><code>wcstol()</code> と <code>wstol()</code> の両関数は、<code>nptr</code> が示すワイド文字列の先頭部分を <code>long int</code> 型の整数表現に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。初めの部分は、<code>iswspace(3C)</code> で識別される空白のワイド文字コードの並びで、空の場合もあります。次が変換対象となる文字コードの並びで、<code>base</code> の値から決定される指数表現の整数として解釈されます。最後の部分は、解釈不能なワイド文字コードの並びで、これには入力文字列の終わりの NULL ワイド文字コードも含まれます。分割後、変換対象となる文字コードの並びを整数に変換しようと試み、その結果を返します。</p> <p><code>base</code> の値が 0 のとき、変換対象となる文字コードの並びの値は 10 進、8 進、または 16 進の定数であるものと期待されます。定数には + または - の符号が付いていてもかまいません。10 進定数は 0 以外の数字で始まり、いくつかの 10 進数で構成されます。8 進定数は接頭辞 0 で認識され、その後には 0 から 7 までの数字をいくつか続けることができます。16 進定数は、接頭辞 0x または 0X で認識され、その後には 0 から 9 までの数字および a (または A) から f (または F) までの文字がいくつか続きます。</p> <p><code>base</code> の値が 2 から 36 までの範囲のとき、変換対象となる文字コードの並びの形式は <code>base</code> が示す基数を用いた整数を表す一連の数字または文字であると期待されます。先頭に + または - の符号が付加されることはありませんが、整数の接尾辞は含まれません。文字 a (または A) から z (または Z) は、それぞれ 10 から 35 までの値に対応します。<code>base</code> の値以内の範囲の文字だけが、数値に対応することができます。<code>base</code> の値が 16 のとき、値を表す一連の数字や文字の前に、0x または 0X のワイド文字コード表現を付加することができます。符号が存在していれば、その後となります。</p> <p>変換対象となる文字コードの並びとは、入力ワイド文字列中の最初の空白以外のワイド文字コードで始まる、期待された形式の、最長のワイド文字コードの並びであると定義できます。入力ワイド文字列が空の場合、または空白を表すワイド文字コードだけからなる場合、または最初の空白以外のワイド文字コードが符号でもなく指定可能な数字や文字でもない場合、変換対象となる文字コードの並びにはワイド文字コードが 1 つも含まれません。</p> <p>変換対象となる文字コードの並びの形式が期待どおりのものであり、<code>base</code> の値が 0 の場合、最初の数字で始まる一連のワイド文字コードは、整数であると解釈されます。変換対象となる文字コードの並びの形式が期待どおりのものであり、<code>base</code> の値が 2 から 36 までの範囲にある場合、その値を基数として変換が行われます。文字に対しては、前述したような方法で数値が割り当てられます。変換対象となる文字コード</p>

の並びが負の符号 (-) で始まっている場合、変換の結果として得られる値は負となります。endptr が示すオブジェクトには、最終部分のワイド文字列へのポインタが書き込まれます。ただし endptr が NULL ポインタの場合を除きます。

ロケールが POSIX 以外の場合、そのシステム固有の他の変換対象となる文字コードの並び形式も使用できます。

変換対象となる文字コードの並びが空の場合、または期待された形式ではない場合、変換処理は行われません。またこのとき、endptr が示すオブジェクトには、nptr の値が書き込まれます。ただし endptr が NULL ポインタの場合を除きます。

watol() 関数は wstol(str, (wchar\_t \*\*)NULL, 10) と同機能です。

watoll() 関数は longlong 型の watol() です。

watoi() 関数は (int)watol() と同機能です。

戻り値 処理が正常に終了すると、wcstol() および wstol() は、変換された値 (もしあれば) を返します。変換が行われなかった場合、0 が返されます。このとき、errno はエラー状態を示す値に設定されることがあります。正しい値が表現可能な値の範囲外であれば、{LONG\_MAX} または {LONG\_MIN} のどちらか (値の符号による) が返され、errno は ERANGE の値に設定されます。

エラー wcstol() および wstol() 関数は、以下の条件のとき異常終了します。

EINVAL base の値はサポートされていない  
 ERANGE 返されるべき値は表現不能である

また wcstol() および wstol() 関数は、以下の状態のときエラーを返します。

EINVAL 変換が実行できない

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 iswalp(3C), iswspace(3C), scanf(3C), wcstod(3C), attributes(5)

注意事項 エラー発生時には 0、{LONG\_MIN}、および {LONG\_MAX} が返されますが、これらは正常終了の場合にも返される値です。したがってエラー発生の有無を知りたいアプリケーションは、errno を 0 に設定して wcstol() または wstol() を呼び出し、処理実行後に errno の値を調べる必要があります。値が 0 以外であれば、エラーが起こったものと判断できます。

代入時に必要であれば、または明示的に指定されていれば、longlong 型から long 型への切り捨てが発生することがあります。

## watoll(3C)

名前	wcstol, wstol, watol, watoll, watoi – ワイド文字列を整数に変換
形式	<pre>#include &lt;wchar.h&gt;  long int <b>wcstol</b>(const wchar_t *nptr, wchar_t **endptr, int base);  #include &lt;widec.h&gt;  long int <b>wstol</b>(const wchar_t *nptr, wchar_t **endptr, int base);  long <b>watol</b>(wchar_t *nptr);  long long <b>watoll</b>(wchar_t *nptr);  int <b>watoi</b>(wchar_t *nptr);</pre>
機能説明	<p><code>wcstol()</code> と <code>wstol()</code> の両関数は、<code>nptr</code> が示すワイド文字列の先頭部分を <code>long int</code> 型の整数表現に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。初めの部分は、<code>iswspace(3C)</code> で識別される空白のワイド文字コードの並びで、空の場合もあります。次が変換対象となる文字コードの並びで、<code>base</code> の値から決定される指数表現の整数として解釈されます。最後の部分は、解釈不能なワイド文字コードの並びで、これには入力文字列の終わりの NULL ワイド文字コードも含まれます。分割後、変換対象となる文字コードの並びを整数に変換しようと試み、その結果を返します。</p> <p><code>base</code> の値が 0 のとき、変換対象となる文字コードの並びの値は 10 進、8 進、または 16 進の定数であるものと期待されます。定数には + または - の符号が付いていてもかまいません。10 進定数は 0 以外の数字で始まり、いくつかの 10 進数で構成されます。8 進定数は接頭辞 0 で認識され、その後には 0 から 7 までの数字をいくつか続けることができます。16 進定数は、接頭辞 0x または 0X で認識され、その後には 0 から 9 までの数字および a (または A) から f (または F) までの文字がいくつか続きます。</p> <p><code>base</code> の値が 2 から 36 までの範囲のとき、変換対象となる文字コードの並びの形式は <code>base</code> が示す基数を用いた整数を表す一連の数字または文字であると期待されます。先頭に + または - の符号が付加されることはありませんが、整数の接尾辞は含まれません。文字 a (または A) から z (または Z) は、それぞれ 10 から 35 までの値に対応します。<code>base</code> の値以内の範囲の文字だけが、数値に対応することができます。<code>base</code> の値が 16 のとき、値を表す一連の数字や文字の前に、0x または 0X のワイド文字コード表現を付加することができます。符号が存在していれば、その後となります。</p> <p>変換対象となる文字コードの並びとは、入力ワイド文字列中の最初の空白以外のワイド文字コードで始まる、期待された形式の、最長のワイド文字コードの並びであると定義できます。入力ワイド文字列が空の場合、または空白を表すワイド文字コードだけからなる場合、または最初の空白以外のワイド文字コードが符号でもなく指定可能な数字や文字でもない場合、変換対象となる文字コードの並びにはワイド文字コードが 1 つも含まれません。</p> <p>変換対象となる文字コードの並びの形式が期待どおりのものであり、<code>base</code> の値が 0 の場合、最初の数字で始まる一連のワイド文字コードは、整数であると解釈されます。変換対象となる文字コードの並びの形式が期待どおりのものであり、<code>base</code> の値が 2 から 36 までの範囲にある場合、その値を基数として変換が行われます。文字に対しては、前述したような方法で数値が割り当てられます。変換対象となる文字コード</p>

の並びが負の符号 (-) で始まっている場合、変換の結果として得られる値は負となります。 *endptr* が示すオブジェクトには、最終部分のワイド文字列へのポインタが書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

ロケールが POSIX 以外の場合、そのシステム固有の他の変換対象となる文字コードの並び形式も使用できます。

変換対象となる文字コードの並びが空の場合、または期待された形式ではない場合、変換処理は行われません。またこのとき、 *endptr* が示すオブジェクトには、 *nptr* の値が書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

*watol()* 関数は *wstol(str, (wchar\_t \*\*)NULL, 10)* と同機能です。

*watoll()* 関数は longlong 型の *watol()* です。

*watoi()* 関数は (int)*watol()* と同機能です。

戻り値 処理が正常に終了すると、 *wcstol()* および *wstol()* は、変換された値 (もしあれば) を返します。変換が行われなかった場合、0 が返されます。このとき、 *errno* はエラー状態を示す値に設定されることがあります。正しい値が表現可能な値の範囲外であれば、 {LONG\_MAX} または {LONG\_MIN} のどれか (値の符号による) が返され、 *errno* は ERANGE の値に設定されます。

エラー *wcstol()* および *wstol()* 関数は、以下の条件のとき異常終了します。

EINVAL *base* の値はサポートされていない

ERANGE 返されるべき値は表現不能である

また *wcstol()* および *wstol()* 関数は、以下の状態のときエラーを返します。

EINVAL 変換が実行できない

属性 次の属性については *attributes(5)* のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 *iswalph(3C)*, *iswspace(3C)*, *scanf(3C)*, *wcstod(3C)*, *attributes(5)*

注意事項 エラー発生時には 0、 {LONG\_MIN}、 および {LONG\_MAX} が返されますが、これらは正常終了の場合にも返される値です。したがってエラー発生の有無を知りたいアプリケーションは、 *errno* を 0 に設定して *wcstol()* または *wstol()* を呼び出し、処理実行後に *errno* の値を調べる必要があります。値が 0 以外であれば、エラーが起こったものと判断できます。

代入時に必要であれば、または明示的に指定されていれば、 longlong 型から long 型への切り捨てが発生することがあります。

## wcscat(3C)

名前	wcstring, wcscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wcscopy, wscopy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsprk, wspbrk, wcsvcs, wcsspn, wssp, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wcscopy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wcschr(const wchar_t *ws, wchat_t wc); wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsvcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscopy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchat_t wc); wchar_t *wsrchr(const wchar_t *ws, wchat_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

### XPG4 と SUS

デフォルトとその  
他の標準仕様



	<pre> size_t wscspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *windex(const wchar_t *ws, wchar_t wc); wchar_t *wrindex(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *wcschr(const wchar_t *ws, wchar_t wc); const wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::wcschr(wchar_t *ws, wchar_t wc); wchar_t *std::wcpbrk(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <i>ws</i>、<i>ws1</i>、および <i>ws2</i> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<i>ws1</i> が指すワイド文字列の終わりに追加します。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<i>ws1</i> と <i>ws2</i> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<i>ws1</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードと、<i>ws2</i> が指す配列の内容とを比較します。<i>ws1</i> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wcscat(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p>wscpy() と wscpy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy() と wcsncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen() と wslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p>wcsrchr() と wsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex() と wrindex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk() と wcspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される <code>wchar_t</code> ポインタを返します。<code>wchar_t</code> ポインタには、<code>wcstok()</code> 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は <code>wcstok()</code> の XPG4 と SUS のバージョンでは利用できません。また、<code>wstok()</code> 関数でも利用できません ( <code>standards(5)</code> 参照 )。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、<code>wcstok()</code> および <code>wstok()</code> は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、<code>wcstok()</code> と <code>wstok()</code> の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき <code>wcstok()</code> と <code>wstok()</code> の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wcscat(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wchr, wchr, wsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wscspn, wscspn, wstok, wstok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);

wchar_t *wchr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);

wchar_t *wchr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchat_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

## wchr(3C)

	<pre> size_t <b>wcspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wcscat()</b> , <b>wscat()</b>	<p>wcscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscmp()</b> , <b>wscmp()</b>	<p>wcscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p>wcsncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wcscpy(), wscpy()</b>	<p><i>wcscpy()</i> と <i>wscpy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複写します。なお NULL のワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p><i>wcslen()</i> と <i>wslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wschr()</b>	<p><i>wcschr()</i> と <i>wschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p><i>wcsrchr()</i> と <i>wsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), wrindex()</b>	<p><i>windex()</i> と <i>wrindex()</i> の両関数の動作は、それぞれ <i>wschr()</i> と <i>wsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wspbrk()</b>	<p><i>wcspbrk()</i> と <i>wspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wcschr(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspn()</b>	<p>wcsspn() と wsspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspn(), wscspn()</b>	<p>wcscspn() と wscspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>



## wcschr(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wcscmp(3C)

名前	wcstring, wscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsprk, wspbrk, wcsvcs, wcsspn, wssp, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wcscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchat_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsvcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchat_t wc); wchar_t *wsrchr(const wchar_t *ws, wchat_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

### XPG4 と SUS

デフォルトとその  
他の標準仕様

	<pre> size_t wscspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *windex(const wchar_t *ws, wchar_t wc); wchar_t *wrindex(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *wcschr(const wchar_t *ws, wchar_t wc); const wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::wcschr(wchar_t *ws, wchar_t wc); wchar_t *std::wcpbrk(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <i>ws</i>、<i>ws1</i>、および <i>ws2</i> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<i>ws1</i> が指すワイド文字列の終わりに追加します。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<i>ws1</i> と <i>ws2</i> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<i>ws1</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードと、<i>ws2</i> が指す配列の内容とを比較します。<i>ws1</i> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wcscmp(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p>wscpy() と wscpy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy() と wcsncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen() と wslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p>wcsrchr() と wsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex() と wrindex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk() と wcspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	<b>wcswcs()</b> 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	<b>wcsspn()</b> と <b>wsspn()</b> の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	<b>wcscspn()</b> と <b>wscspn()</b> の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	<b>wcstok()</b> と <b>wstok()</b> を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される <code>wchar_t</code> ポインタを返します。<code>wchar_t</code> ポインタには、<b>wcstok()</b> 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は <b>wcstok()</b> の XPG4 と SUS のバージョンでは利用できません。また、<b>wstok()</b> 関数でも利用できません ( <a href="#">standards(5)</a> 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、<b>wcstok()</b> および <b>wstok()</b> は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、<b>wcstok()</b> と <b>wstok()</b> の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき <b>wcstok()</b> と <b>wstok()</b> の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wscmp(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前	wscoll, wscoll – 照合情報を使ってワイド文字列を比較						
形式	<pre>#include &lt;wchar.h&gt;  int <b>wscoll</b>(const wchar_t *ws1, const wchar_t *ws2); int <b>wscoll</b>(const wchar_t *ws1, const wchar_t *ws2);</pre>						
機能説明	wscoll() と wscoll() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両文字列は、現ロケールの LC_COLLATE カテゴリに従って解釈されます。						
戻り値	比較処理が正常に終了すると、wscoll() および wscoll() はその結果を示す値を返します。現ロケールに従って比較した結果、両文字列が一致していれば 0 を、ws1 が指す文字列の方が ws2 が指す文字列より大きければ 0 より大きい値を、小さければ 0 より小さい値をそれぞれ返します。エラーが発生した場合、wscoll() および wscoll() は errno を設定することがありますが、エラー発生を表す戻り値は定義されていません。						
エラー	<p>wscoll() および wscoll() 関数は、以下の状態のときエラーを返します。</p> <p>EINVAL            引数 ws1 または ws2 が示すワイド文字列に、照合手順に定義されていないワイド文字コードが含まれていた</p> <p>ENOSYS            関数はサポートされていない</p>						
使用法	<p>エラーを示す戻り値は定義されていないため、エラー発生の有無を知りたいアプリケーションは、errno を 0 に設定して wscoll() または wscoll() を呼び出し、処理実行後に errno の値を調べる必要があります。値が 0 以外であれば、エラーが起こったものと判断できます。</p> <p>大量のデータを分類するときは、wcsxfrm(3C) および wcscmp(3C) 関数を使用してください。</p> <p>wscoll() と wscoll() 関数は、setlocale(3C) を呼び出してロケールを変更しない限り、マルチスレッドのアプリケーションで安全に使用できます。</p>						
属性	<p>次の属性については attributes(5) のマニュアルページを参照してください。</p> <table border="1"> <thead> <tr> <th>属性タイプ</th> <th>属性値</th> </tr> </thead> <tbody> <tr> <td>MT レベル</td> <td>例外付き MT-safe</td> </tr> <tr> <td>CSI</td> <td>対応済み</td> </tr> </tbody> </table>	属性タイプ	属性値	MT レベル	例外付き MT-safe	CSI	対応済み
属性タイプ	属性値						
MT レベル	例外付き MT-safe						
CSI	対応済み						
関連項目	setlocale(3C), wcscmp(3C), wcsxfrm(3C), attributes(5)						

## wcscpy(3C)

名前	wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wcsvcs, wcsspn, wssp, wcsspn, wcsspn, wctok, wctok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wcscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wcschr(const wchar_t *ws, wchar_t wc); wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsvcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wctok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wctok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

### XPG4 と SUS

デフォルトとその  
他の標準仕様



	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <code>ws</code>、<code>ws1</code>、および <code>ws2</code> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<code>ws2</code> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<code>ws1</code> が指すワイド文字列の終わりに追加します。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<code>ws2</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードを、<code>ws1</code> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<code>ws1</code> と <code>ws2</code> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<code>ws1</code> の指すワイド文字列中のコードが <code>ws2</code> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<code>ws1</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードと、<code>ws2</code> が指す配列の内容とを比較します。<code>ws1</code> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wcscpy(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wcscpy(), wscpy()</b>	<p>wcscpy()とwscpy()の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wsnncpy()</b>	<p>wcsncpy()とwsncpy()の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen()とwslen()の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wschr()</b>	<p>wcschr()とwschr()の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p>wcsrchr()とwsrchr()の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex()とwrindex()の両関数の動作は、それぞれwschr()とwsrchr()の動作と同じです。</p>
<b>wcspbrk(), wspbrk()</b>	<p>wcspbrk()とwspbrk()の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される <code>wchar_t</code> ポインタを返します。<code>wchar_t</code> ポインタには、<code>wcstok()</code> 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は <code>wcstok()</code> の XPG4 と SUS のバージョンでは利用できません。また、<code>wstok()</code> 関数でも利用できません ( <code>standards(5)</code> 参照 )。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、<code>wcstok()</code> および <code>wstok()</code> は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、<code>wcstok()</code> と <code>wstok()</code> の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき <code>wcstok()</code> と <code>wstok()</code> の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wcscpy(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wcschr(const wchar_t *ws, wchat_t wc);
wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchat_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

## wcscspn(3C)

	<pre> size_t <b>wcscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wcscat()</b> , <b>wscat()</b>	<p>wcscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscmp()</b> , <b>wscmp()</b>	<p>wcscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p>wcsncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscopy()</b>	<p><i>wscpy()</i> と <i>wscopy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複製します。なお NULL のワイド文字コードに続くワイド文字コードは、複製の対象とはなりません。重なりあうオブジェクト間で複製が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wcslen()</b>	<p><i>wcslen()</i> と <i>wcslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p><i>wcschr()</i> と <i>wcschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p><i>wcsrchr()</i> と <i>wcsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), windex()</b>	<p><i>windex()</i> と <i>windex()</i> の両関数の動作は、それぞれ <i>wcschr()</i> と <i>wcsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p><i>wcspbrk()</i> と <i>wcspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wcscspn(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspn()</b>	<p>wcsspn() と wsspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspn(), wcscspn()</b>	<p>wcscspn() と wcscspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なってもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>



## wscspn(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から 上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wcsetno(3C)

名前 cset, csetlen, csetcol, csetno, wcsetno – EUC コードセットに関する情報の取得

形式

```
#include <euc.h>

int csetlen(int codeset);

int csetcol(int codeset);

int csetno(unsigned char c);

#include <widec.h>

int wcsetno(wchar_t pc);
```

機能説明 csetlen() と csetcol() は、ともにコードセット番号 *codeset* を取得します。コードセット番号は 0、1、2、または 3 を指定してください。csetlen() 関数は、所定の拡張 UNIX コード (EUC) コードセットの文字 (コードセット 2 およびコードセット 3 用のシングルシフト文字 SS2 および SS3 を除く) を表現するのに必要なバイト数を返します。csetcol() 関数は、所定の EUC コードセットの文字がディスプレイ上で占めるカラム数を返します。

csetno() 関数は、最初のバイトが *c* である EUC 文字に対するコードセット番号 (0、1、2、または 3) を返すマクロです。たとえば、

```
#include<euc.h> . . . x+=csetcol(csetno(c));
```

によって、カウンタ *x* (カーソル位置など) は、最初のバイトが *c* の文字幅分増加します。

wcsetno() 関数は、所定のワイド文字 *pc* に対するコードセット番号 (0、1、2、または 3) を返すマクロとして定義されています。たとえば、

```
#include<euc.h> #include<widec.h> . . . x+=csetcol(wcsetno(pc));
```

によって、カウンタ *x* (カーソル位置など) は、ワイド文字 *pc* の幅だけ増加します。

使用法 cset(), csetlen(), csetcol(), csetno() または wcsetno() 関数は setlocale(3C) がロケールの変更のために呼び出されていない限り、マルチスレッドアプリケーション上で安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe

関連項目 setlocale(3C) euclen(3C), attributes(5)

名前	wcsftime – 日付および時刻をワイド文字列に変換						
形式	#include <wchar.h>						
<b>XPG4 と SUS</b>	size_t <b>wcsftime</b> (wchar_t *wcs, size_t maxsize, const char *format, const struct tm *timptr);						
デフォルトとその他の標準仕様	size_t <b>wcsftime</b> (wchar_t *wcs, size_t maxsize, const wchar_t *format, const struct tm *timptr);						
機能説明	<p>wcsftime() 関数は、次の点を除いて strftime(3C) 関数と同じです。</p> <ul style="list-style-type: none"> <li>■ <i>wcs</i> 引数は、生成される出力が置かれる ワイド文字の配列の初期エレメントを指す</li> <li>■ <i>maxsize</i> 引数は、出力の配列に格納するワイド文字の最大数を表す</li> <li>■ <i>format</i> 引数はワイド文字の文字列を表し、変換に指定すると対応するワイド文字の並びに置換される</li> <li>■ 戻り値は、出力の配列に格納したワイド文字コード数を表示する</li> </ul> <p>重なり合うオブジェクト間でコピー処理が発生した場合の動作結果は未定義です。</p>						
戻り値	<p>結果として生成されたワイド文字コード数 (終端の NULL ワイド文字を含む) が <i>maxsize</i> 以下の場合、wcsftime() は、<i>wcs</i> が指す配列に格納したワイド文字コード数 (終端の NULL ワイド文字は含まない) を返します。数が <i>maxsize</i> を超えていた場合には、0 が返され、配列の内容は未定義となります。</p> <p>wcftime() 関数は malloc(3C) を使用します。malloc() が異常終了すると、errno が malloc() により設定されます。</p>						
属性	次の属性については attributes(5) のマニュアルページを参照してください。						
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">属性タイプ</th> <th style="text-align: center;">属性値</th> </tr> </thead> <tbody> <tr> <td>MT レベル</td> <td>例外付き MT-Safe</td> </tr> <tr> <td>CSI</td> <td>対応済み</td> </tr> </tbody> </table>	属性タイプ	属性値	MT レベル	例外付き MT-Safe	CSI	対応済み
属性タイプ	属性値						
MT レベル	例外付き MT-Safe						
CSI	対応済み						
関連項目	malloc(3C), setlocale(3C), strftime(3C), attributes(5), standards(5)						
注意事項	wcsftime() 関数は、setlocale(3C) がロケールの変更で呼び出されない限り、マルチスレッドのアプリケーションで安全に使用できます。						

## wcslen(3C)

名前	wcstring, wcsat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wcsspn, wcsspn, wstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

### XPG4 と SUS

デフォルトとその  
他の標準仕様

	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <code>ws</code>、<code>ws1</code>、および <code>ws2</code> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<code>ws2</code> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<code>ws1</code> が指すワイド文字列の終わりに追加します。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<code>ws2</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードを、<code>ws1</code> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<code>ws1</code> と <code>ws2</code> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<code>ws1</code> の指すワイド文字列中のコードが <code>ws2</code> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<code>ws1</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードと、<code>ws2</code> が指す配列の内容とを比較します。<code>ws1</code> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wcslen(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p>wscpy() と wscpy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy() と wcsncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen() と wslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p>wcsrchr() と wsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex() と wrindex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk() と wcspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wcslen(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`



名前 | wcstring, wscat, wscat, wcsnecat, wsnecat, wscmp, wscmp, wcsncmp, wsnncmp, wscpy, wscpy, wcsncpy, wsnncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wscswcs, wcsspn, wssp, wscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsnecat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wscswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsnecat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsnncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchat_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

## wscnecat(3C)

	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p>wscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscnecat()</b> , <b>wsnecat()</b>	<p>wscnecat() と wsnecat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p>wscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wscncmp()</b> , <b>wsncmp()</b>	<p>wscncmp() と wscncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscncpy(), wscpy()</b>	<p><i>wscncpy()</i> と <i>wscpy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複写します。なお NULL のワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p><i>wcslen()</i> と <i>wslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wschr()</b>	<p><i>wcschr()</i> と <i>wschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p><i>wcsrchr()</i> と <i>wsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), wrindex()</b>	<p><i>windex()</i> と <i>wrindex()</i> の両関数の動作は、それぞれ <i>wschr()</i> と <i>wsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wspbrk()</b>	<p><i>wcspbrk()</i> と <i>wspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wcsncat(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspnl()</b>	<p>wcsspn() と wsspnl() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コードだけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspnl(), wscspnl()</b>	<p>wcscspnl() と wscspnl() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なってもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

### wcsncat(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wcsncmp(3C)

名前	wcstring, wcsconcat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wcsspn, wcsspn, wstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

XPG4 と SUS  
デフォルトとその  
他の標準仕様

	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <code>ws</code>、<code>ws1</code>、および <code>ws2</code> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<code>ws2</code> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<code>ws1</code> が指すワイド文字列の終わりに追加します。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<code>ws2</code> が指す配列中の <math>n</math> 個以下の数のワイド文字コードを、<code>ws1</code> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<code>ws1</code> と <code>ws2</code> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<code>ws1</code> の指すワイド文字列中のコードが <code>ws2</code> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<code>ws1</code> が指す配列中の <math>n</math> 個以下の数のワイド文字コードと、<code>ws2</code> が指す配列の内容とを比較します。<code>ws1</code> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wcsncmp(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p>wscpy() と wscpy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy() と wcsncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen() と wslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p>wcsrchr() と wcsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex() と wrindex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk() と wcspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>



<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wcsncmp(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wscspn, wscspn, wctok, wtok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wcschr(const wchar_t *ws, wchat_t wc);
wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wctok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wctok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchat_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

## wscncpy(3C)

	<pre> size_t wscspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *windex(const wchar_t *ws, wchar_t wc); wchar_t *wrindex(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *wcschr(const wchar_t *ws, wchar_t wc); const wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::wcschr(wchar_t *ws, wchar_t wc); wchar_t *std::wcpbrk(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
wscat(), wscat()	<p>wscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
wcsncat(), wcsncat()	<p>wcsncat() と wcsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
wscmp(), wscmp()	<p>wscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
wscncmp(), wscncmp()	<p>wscncmp() と wscncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscopy(), wscopy()</b>	<p><i>wscopy()</i> と <i>wscopy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複製します。なお NULL のワイド文字コードに続くワイド文字コードは、複製の対象とはなりません。重なりあうオブジェクト間で複製が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wcslen()</b>	<p><i>wcslen()</i> と <i>wcslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p><i>wcschr()</i> と <i>wcschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p><i>wcsrchr()</i> と <i>wcsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), windex()</b>	<p><i>windex()</i> と <i>windex()</i> の両関数の動作は、それぞれ <i>wcschr()</i> と <i>wcsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p><i>wcspbrk()</i> と <i>wcspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wcsncpy(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspn()</b>	<p>wcsspn() と wsspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コードだけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspn(), wscspn()</b>	<p>wcscspn() と wscspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wcsncpy(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wcspbrk(3C)

名前	wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcsvcs, wcsspn, wssp, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchat_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsvcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchat_t wc); wchar_t *wsrchr(const wchar_t *ws, wchat_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

### XPG4 と SUS

デフォルトとその  
他の標準仕様



	<pre> size_t <b>wcspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <code>ws</code>、<code>ws1</code>、および <code>ws2</code> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wcscat()</b> , <b>wscat()</b>	<p><code>wcscat()</code> と <code>wscat()</code> の両関数は、<code>ws2</code> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<code>ws1</code> が指すワイド文字列の終わりに追加します。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<code>ws2</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードを、<code>ws1</code> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscmp()</b> , <b>wscmp()</b>	<p><code>wcscmp()</code> と <code>wscmp()</code> の両関数は、<code>ws1</code> と <code>ws2</code> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<code>ws1</code> の指すワイド文字列中のコードが <code>ws2</code> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<code>ws1</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードと、<code>ws2</code> が指す配列の内容とを比較します。<code>ws1</code> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wcspbrk(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wcscpy(), wscpy()</b>	<p>wcscpy()とwscpy()の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy()とwcsncpy()の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen()とwslen()の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr()とwcschr()の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p>wcsrchr()とwcsrchr()の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex()とwrindex()の両関数の動作は、それぞれwcschr()とwcsrchr()の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk()とwcspbrk()の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される <code>wchar_t</code> ポインタを返します。<code>wchar_t</code> ポインタには、<code>wcstok()</code> 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は <code>wcstok()</code> の XPG4 と SUS のバージョンでは利用できません。また、<code>wstok()</code> 関数でも利用できません ( <code>standards(5)</code> 参照 )。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、<code>wcstok()</code> および <code>wstok()</code> は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、<code>wcstok()</code> と <code>wstok()</code> の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき <code>wcstok()</code> と <code>wstok()</code> の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wcspbrk(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wcschr(const wchar_t *ws, wchar_t wc);
wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchar_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

## wcsrchr(3C)

	<pre> size_t <b>wcspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wcscat()</b> , <b>wscat()</b>	<p>wcscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscmp()</b> , <b>wscmp()</b>	<p>wcscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p>wcsncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscopy()</b>	<p><i>wscpy()</i> と <i>wscopy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複製します。なお NULL のワイド文字コードに続くワイド文字コードは、複製の対象とはなりません。重なりあうオブジェクト間で複製が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p><i>wcslen()</i> と <i>wslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wschr()</b>	<p><i>wcschr()</i> と <i>wschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p><i>wcsrchr()</i> と <i>wsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), wrindex()</b>	<p><i>windex()</i> と <i>wrindex()</i> の両関数の動作は、それぞれ <i>wschr()</i> と <i>wsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wspbrk()</b>	<p><i>wcspbrk()</i> と <i>wspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wcsrchr(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspnl()</b>	<p>wcsspn() と wsspnl() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コードだけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspnl(), wscspnl()</b>	<p>wcscspnl() と wscspnl() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なってもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>



## wcsrchr(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wcsspn(3C)

名前	wcstring, wcsconcat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsprk, wspbrk, wswcs, wcsspn, wssp, wcspsn, wcsspn, wstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcspsn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcspsn(const wchar_t *ws1, const wchar_t *ws2);</pre>

XPG4 と SUS  
デフォルトとその  
他の標準仕様

	<pre> size_t <b>wcssp</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <code>ws</code>、<code>ws1</code>、および <code>ws2</code> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wcscat()</b> , <b>wscat()</b>	<p><code>wcscat()</code> と <code>wscat()</code> の両関数は、<code>ws2</code> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<code>ws1</code> が指すワイド文字列の終わりに追加します。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<code>ws2</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードを、<code>ws1</code> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscmp()</b> , <b>wscmp()</b>	<p><code>wcscmp()</code> と <code>wscmp()</code> の両関数は、<code>ws1</code> と <code>ws2</code> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<code>ws1</code> の指すワイド文字列中のコードが <code>ws2</code> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<code>ws1</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードと、<code>ws2</code> が指す配列の内容とを比較します。<code>ws1</code> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wcsspn(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscspy(), wscopy()</b>	<p>wscspy() と wscopy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wsncpy()</b>	<p>wcsncpy() と wsncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen() と wslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wschr()</b>	<p>wcschr() と wschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p>wcsrchr() と wsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex() と wrindex() の両関数の動作は、それぞれwschr() と wsrchr() の動作と同じです。</p>
<b>wcspbrk(), wspbrk()</b>	<p>wcspbrk() と wspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	<b>wcswcs()</b> 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcssp(), wssp()</b>	<b>wcssp()</b> と <b>wssp()</b> の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	<b>wcscspn()</b> と <b>wscspn()</b> の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	<b>wcstok()</b> と <b>wstok()</b> を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される <code>wchar_t</code> ポインタを返します。<code>wchar_t</code> ポインタには、<b>wcstok()</b> 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は <b>wcstok()</b> の XPG4 と SUS のバージョンでは利用できません。また、<b>wstok()</b> 関数でも利用できません ( <a href="#">standards(5)</a> 参照 )。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、<b>wcstok()</b> および <b>wstok()</b> は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、<b>wcstok()</b> と <b>wstok()</b> の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき <b>wcstok()</b> と <b>wstok()</b> の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wcsspn(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 wcstod, wstod, watof – ワイド文字列を倍精度浮動小数点数に変換

形式 

```
#include <wchar.h>

double wcstod(const wchar_t *nptr, wchar_t **endptr);
double wstod(const wchar_t *nptr, wchar_t **endptr);
double watof(wchar_t *nptr);
```

機能説明 wcstod() と wstod() の両関数は、*nptr* が示すワイド文字列の先頭部分を double 型の表現に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。初めの部分は、iswspace(3C) で識別される空白のワイド文字コードの並びで、空の場合もあります。次が変換対象となる文字コードの並びで、浮動小数点数として解釈されます。最後の部分は、解釈不能なワイド文字コードの並びで、これには入力文字列の終わりの NULL ワイド文字コードも含まれます。分割後、変換対象となる文字コードの並びを浮動小数点数に変換しようと試み、その結果を返します。

変換対象となる文字コードの並びとして期待される形式は、先頭が + または - の符号 (なくても可) で、その後が小数点を含むまたは含まない 1 つ以上の数字の集まりで、必要であれば最後に指数部が続きます。指数部は、文字 e または E で、その後にはオブションとして符号やいくつかの 10 進数を付加することもできます。変換対象となる文字コードの並びとは、入力ワイド文字列中の最初の空白でないワイド文字コードで始まる、期待された形式の、最長のワイド文字コードの並びであると定義できます。入力ワイド文字列が空の場合、または空白を表すワイド文字コードだけからなる場合、または最初の空白でないワイド文字コードが符号でも数字でも小数点でもない場合、変換対象となる文字コードの並びにはワイド文字コードが 1 つも含まれません。

変換対象となる文字コードの並びの形式が期待どおりの場合、最初の数字または小数点 (どちらか最初に現れた方) で始まる一連のワイド文字コードは、C 言語で定義される浮動小数点数であると解釈されます。C 言語の定義と異なる点は、ピリオドの代わりに小数点が使われること、指数も小数点も現れなければワイド文字列中の最後の数字の後に小数点が続くと見なされることです。変換対象となる文字コードの並びが負の符号 (-) で始まっている場合、変換の結果として得られた値は負となります。*endptr* が示すオブジェクトには、最終部分のワイド文字列へのポインタが書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

小数点はプログラムのロケール (LC\_NUMERIC カテゴリ) 中に定義されます。POSIX ロケール、および小数点が定義されていないロケールにおいては、小数点のデフォルト値はピリオド (.) となります。

ロケールが POSIX 以外の場合、そのシステム固有の他の変換対象となる文字コードの並び形式も使用できます。

変換対象となる文字コードの並びが空の場合、または期待された形式ではない場合、変換処理は行われません。またこのとき、*endptr* が示すオブジェクトには、*nptr* の値が書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

*watof(str)* 関数は *wstod(str, (wchar\_t \*\*)NULL)* と同じです。

## wcstod(3C)

**戻り値** `wcstod()` および `wstod()` 関数は、変換された値 (もしあれば) を返します。変換が行われなかった場合、0 が返されます。このとき、`errno` は `EINVAL` の値に設定されることがあります。

正しい値が表現可能な値の範囲外であれば、`±HUGE_VAL` のどれか (値の符号による) が返され、`errno` は `ERANGE` の値に設定されます。

正しい値がアンダフローを起こしてしまうのであれば、0 が返され、`errno` は `ERANGE` の値に設定されます。

**エラー** `wcstod()` および `wstod()` 関数は、以下の状態のときエラーを返します。

`ERANGE` 返されるべき値が、オーバーフローまたはアンダフローを起こしてしまう

`wcstod()` および `wcstod()` 関数は、以下の状態のときエラーを返します。

`EINVAL` 変換が実行できない

**使用法** エラー発生時には 0 が返されますが、これは正常終了の場合にも返される値です。したがってエラー発生の有無を知りたいアプリケーションは、`errno` を 0 に設定して `wcstod()` または `wstod()` を呼び出し、処理実行後に `errno` の値を調べる必要があります。値が 0 以外であれば、エラーが起こったものと判断できます。

**属性** 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

**関連項目** `iswspace(3C)`, `localeconv(3C)`, `scanf(3C)`, `setlocale(3C)`, `wcstol(3C)`, `attributes(5)`



名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wcsvcs, wcsspn, wssp, wscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wcschr(const wchar_t *ws, wchat_t wc);
wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsvcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchat_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

wcstok(3C)

	<pre> size_t <b>wcspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wcscat()</b> , <b>wscat()</b>	<p>wcscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscmp()</b> , <b>wscmp()</b>	<p>wcscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p>wcsncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wcscpy(), wscpy()</b>	<p><i>wcscpy()</i> と <i>wscpy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複製します。なお NULL のワイド文字コードに続くワイド文字コードは、複製の対象とはなりません。重なりあうオブジェクト間で複製が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p><i>wcslen()</i> と <i>wslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p><i>wcschr()</i> と <i>wcschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p><i>wcsrchr()</i> と <i>wsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), windex()</b>	<p><i>windex()</i> と <i>windex()</i> の両関数の動作は、それぞれ <i>wcschr()</i> と <i>wcsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p><i>wcspbrk()</i> と <i>wcspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wcstok(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspn()</b>	<p>wcsspn() と wsspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コードだけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspn(), wscspn()</b>	<p>wcscspn() と wscspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なってもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wcstok(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wcstol(3C)

名前	wcstol, wstol, watol, watoll, watoi – ワイド文字列を整数に変換
形式	<pre>#include &lt;wchar.h&gt;  long int <b>wcstol</b>(const wchar_t *nptr, wchar_t **endptr, int base);  #include &lt;widec.h&gt;  long int <b>wstol</b>(const wchar_t *nptr, wchar_t **endptr, int base);  long <b>watol</b>(wchar_t *nptr);  long long <b>watoll</b>(wchar_t *nptr);  int <b>watoi</b>(wchar_t *nptr);</pre>
機能説明	<p><code>wcstol()</code> と <code>wstol()</code> の両関数は、<code>nptr</code> が示すワイド文字列の先頭部分を <code>long int</code> 型の整数表現に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。初めの部分は、<code>iswspace(3C)</code> で識別される空白のワイド文字コードの並びで、空の場合もあります。次が変換対象となる文字コードの並びで、<code>base</code> の値から決定される指数表現の整数として解釈されます。最後の部分は、解釈不能なワイド文字コードの並びで、これには入力文字列の終わりの NULL ワイド文字コードも含まれます。分割後、変換対象となる文字コードの並びを整数に変換しようと試み、その結果を返します。</p> <p><code>base</code> の値が 0 のとき、変換対象となる文字コードの並びの値は 10 進、8 進、または 16 進の定数であるものと期待されます。定数には + または - の符号が付いていてもかまいません。10 進定数は 0 以外の数字で始まり、いくつかの 10 進数で構成されます。8 進定数は接頭辞 0 で認識され、その後には 0 から 7 までの数字をいくつか続けることができます。16 進定数は、接頭辞 0x または 0X で認識され、その後には 0 から 9 までの数字および a (または A) から f (または F) までの文字がいくつか続きます。</p> <p><code>base</code> の値が 2 から 36 までの範囲のとき、変換対象となる文字コードの並びの形式は <code>base</code> が示す基数を用いた整数を表す一連の数字または文字であると期待されます。先頭に + または - の符号が付加されることはありませんが、整数の接尾辞は含まれません。文字 a (または A) から z (または Z) は、それぞれ 10 から 35 までの値に対応します。<code>base</code> の値以内の範囲の文字だけが、数値に対応することができます。<code>base</code> の値が 16 のとき、値を表す一連の数字や文字の前に、0x または 0X のワイド文字コード表現を付加することができます。符号が存在していれば、その後となります。</p> <p>変換対象となる文字コードの並びとは、入力ワイド文字列中の最初の空白以外のワイド文字コードで始まる、期待された形式の、最長のワイド文字コードの並びであると定義できます。入力ワイド文字列が空の場合、または空白を表すワイド文字コードだけからなる場合、または最初の空白以外のワイド文字コードが符号でもなく指定可能な数字や文字でもない場合、変換対象となる文字コードの並びにはワイド文字コードが 1 つも含まれません。</p> <p>変換対象となる文字コードの並びの形式が期待どおりのものであり、<code>base</code> の値が 0 の場合、最初の数字で始まる一連のワイド文字コードは、整数であると解釈されます。変換対象となる文字コードの並びの形式が期待どおりのものであり、<code>base</code> の値が 2 から 36 までの範囲にある場合、その値を基数として変換が行われます。文字に対しては、前述したような方法で数値が割り当てられます。変換対象となる文字コード</p>

の並びが負の符号 (-) で始まっている場合、変換の結果として得られる値は負となります。 *endptr* が示すオブジェクトには、最終部分のワイド文字列へのポインタが書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

ロケールが POSIX 以外の場合、そのシステム固有の他の変換対象となる文字コードの並び形式も使用できます。

変換対象となる文字コードの並びが空の場合、または期待された形式ではない場合、変換処理は行われません。またこのとき、 *endptr* が示すオブジェクトには、 *nptr* の値が書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

*watol*( ) 関数は *wstol*(*str*, (*wchar\_t* \*\*)NULL, 10) と同機能です。

*watoll*( ) 関数は longlong 型の *watol*( ) です。

*watoi*( ) 関数は (int)*watol*( ) と同機能です。

戻り値 処理が正常に終了すると、 *wcstol*( ) および *wstol*( ) は、変換された値 (もしあれば) を返します。変換が行われなかった場合、0 が返されます。このとき、 *errno* はエラー状態を示す値に設定されることがあります。正しい値が表現可能な値の範囲外であれば、 {LONG\_MAX} または {LONG\_MIN} のどちらか (値の符号による) が返され、 *errno* は ERANGE の値に設定されます。

エラー *wcstol*( ) および *wstol*( ) 関数は、以下の条件のとき異常終了します。

EINVAL *base* の値はサポートされていない

ERANGE 返されるべき値は表現不能である

また *wcstol*( ) および *wstol*( ) 関数は、以下の状態のときエラーを返します。

EINVAL 変換が実行できない

属性 次の属性については *attributes*(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 *iswalph*(3C), *iswspace*(3C), *scanf*(3C), *wcstod*(3C), *attributes*(5)

注意事項 エラー発生時には 0、 {LONG\_MIN}、 および {LONG\_MAX} が返されますが、これらは正常終了の場合にも返される値です。したがってエラー発生の有無を知りたいアプリケーションは、 *errno* を 0 に設定して *wcstol*( ) または *wstol*( ) を呼び出し、処理実行後に *errno* の値を調べる必要があります。値が 0 以外であれば、エラーが起こったものと判断できます。

代入時に必要であれば、または明示的に指定されていれば、 longlong 型から long 型への切り捨てが発生することがあります。

## wcstombs(3C)

名前 wcstombs – ワイド文字列を文字列に変換

形式 `#include <stdlib.h>`

```
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);
```

機能説明 `wcstombs()` 関数は、`pwcs` が示す配列からのワイド文字コードのシーケンスを、複数バイト文字のシーケンスに変換し、これらの複数バイト文字を `s` が示す配列に格納します。この場合、複数バイト文字が合計 `n` バイトの限度を超えるか、NULL バイトが格納されると停止します。それぞれのワイド文字コードは `wctomb(3C)` への呼び出しによる変換と同じように変換されます。

この関数の動作は、現在のロケールの `LC_CTYPE` カテゴリに影響を受けます。

`n` バイト以下の場合、`s` によって示された配列に修正されます。コピーが、オーバーラップするオブジェクト間で行われた場合には、動作は定義されません。`s` が NULL ポインタである場合、`wcstombs()` は、`n` の値にかかわらず、配列全体を変換するために必要な長さを返します。格納される値はありません。

戻り値 有効な複数バイト文字に対応していないワイド文字コードが検出されると、`wcstombs()` は `(size_t)-1` を返します。それ以外の場合、`wcstombs()` は終了 NULL バイトを数えずに、格納されたバイトの数を返します。戻り値が `n` の場合、配列は NULL 終了にはなりません。

エラー `wcstombs()` 関数は以下の状態の時に失敗することがあります。

EILSEC                   ワイド文字コードが有効な複数バイト文字に対応していない

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe
CSI	対応済み

関連項目 `mblen(3C)`, `mbstowcs(3C)`, `mbtowc(3C)`, `setlocale(3C)`, `wctomb(3C)`, `attributes(5)`



名前 wcstoul – ワイド文字列を符号なしロング整数に変換

形式 #include <wchar.h>

```
unsigned long int wcstoul(const wchar_t *nptr, wchar_t **endptr, int
    base);
```

機能説明

wcstoul() 関数は、*nptr* が示すワイド文字列の先頭部分を unsigned long int つまり符号なしロング整数表現に変換します。まず、指定されたワイド文字列を3つの部分に分解します。初めの部分は、iswspace(3C) で規定される空白のワイド文字コードの並びで、空の場合もあります。次が変換対象となる文字コードの並びで、*base* の値で指定された基数で表される整数として解釈されます。最後の部分は、解釈不能なワイド文字コードのシーケンスで、これには入力文字列の終わりの NULL ワイド文字コードも含まれます。分割後、変換対象となる文字コードの並びを符号なし整数に変換しようと試み、その結果を返します。

*base* の値が 0 のとき、変換対象となる文字コードの並びの値は 10 進、8 進、または 16 進の定数であると予想されます。定数には + または - の符号が付くこともあります。10 進定数は 0 以外の数で始まり、いくつかの 10 進数で構成されます。8 進定数は接頭辞 0 で始まり、その後には 0 から 7 までの数字がいくつか続きます。16 進定数は、接頭辞 0x または 0X で始まり、その後には 0 から 9 までの数字および a (または A) から f (または F) までの文字 (数値 10 から 15 に対応) がいくつか続きます。

*base* の値が 2 から 36 までの範囲のとき、変換対象となる文字コードの並びの形式は *base* が示す基数に基づいた整数を表す一連の数字または文字であると予想されます。先頭に + または - の符号が付加されることはありますが、整数の接尾辞は含まれません。文字 a (または A) から z (または Z) は、それぞれ 10 から 35 までの値に対応します。*base* の値以内の範囲にある文字だけが、数値に対応することができます。*base* の値が 16 のとき、値を表す一連の数字や文字の前に、0x または 0X のワイド文字コード表現を付加することができます。符号が存在していれば、そのあとに付加します。

変換対象となる文字コードの並びとは、入力ワイド文字列中の最初の空白以外のワイド文字コードで始まる、予想された形式の、最長のワイド文字コード群であると定義できます。入力ワイド文字列が空の場合、または空白を表すワイド文字コードだけからなる場合、または最初の空白以外のワイド文字コードが符号でもなく指定可能な数字や文字でもない場合、変換対象となる文字コードの並びにはワイド文字コードが 1 つも含まれません。

変換対象となる文字コードの並びが予想どおりの形式であり、*base* の値が 0 の場合、最初の数字で始まる一連のワイド文字コードは、整数であると解釈されます。変換対象となる文字コードの並びが予想どおりの形式であり、*base* の値が 2 から 36 までの範囲にある場合、その値を基数として変換が行われます。文字に対しては、前述したような方法で数値が割り当てられます。変換対象となる文字コードの並びが負の符号 (-) で始まっている場合、変換の結果として得られた値は負となります。*endptr* が示すオブジェクトには、最終のワイド文字列へのポインタが書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

ロケールが POSIX 以外の場合、変換対象となる文字コードの並びの形式も使用できません。

## wcstoul(3C)

変換対象となる文字コードの並びが空の場合、または予想された形式ではない場合、変換処理は行われません。またこのとき、*endptr* が示すオブジェクトには、*nptr* の値が書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

エラー発生時には 0 および `ULONG_MAX` が返されますが、0 は正常終了の場合にも返される値です。したがってエラー発生の有無を知りたいアプリケーションは、`errno` を 0 に設定して `wcstoul()` を呼び出し、処理実行後に `errno` の値をチェックする必要があります。値が 0 でなければ、エラーが起こったものと判断できません。

**戻り値** 処理が正常に終了すると、`wcstoul()` は、変換された値 (もしあれば) を返します。`errno` の設定は変更しません。変換が行われなかった場合、0 が返されます。このとき、`errno` はエラー発生を示す値に設定されることがあります。正しい値が表現可能な値の範囲外であれば、`ULONG_MAX` が返され、`errno` は `ERANGE` の値に設定されます。

**エラー** `wcstoul()` 関数は、以下の条件のとき異常終了します。

`EINVAL` *base* の値はサポートされていない

`ERANGE` 返されるべき値は表現不能である

また `wcstoul()` は、以下の条件のとき異常終了する場合があります。

`EINVAL` 変換が実行できない

**使用法** `wcstod(3C)` や `wcstol(3C)` とは異なり、`wcstoul()` は常に負ではない数を返す必要があります。そのため `wcstoul()` の戻り値を `wcstoul()` の範囲外数値として使用します。`wcstoul()` は、範囲外数値が常に負の場合、正確性を失うだけでなく、さらに深刻な問題を生じることがあります。

**属性** 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

**関連項目** `isspace(3C)`, `iswalph(3C)`, `scanf(3C)`, `wcstod(3C)`, `wcstol(3C)`, `attributes(5)`

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wcsvcs, wcsspn, wssp, wscspn, wscspn, wctok, wctok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wcschr(const wchar_t *ws, wchar_t wc);
wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsvcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wctok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wctok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <wchar.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchar_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

## wcstring(3C)

	<pre> size_t <b>wcspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wcscat()</b> , <b>wscat()</b>	<p>wcscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscmp()</b> , <b>wscmp()</b>	<p>wcscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p>wcsncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wcscpy(), wscpy()</b>	<p><b>wcscpy()</b> と <b>wscpy()</b> の両関数は、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><b>wcsncpy()</b> と <b>wcsncpy()</b> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複写します。なお NULL のワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p><b>wcslen()</b> と <b>wslen()</b> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p><b>wcschr()</b> と <b>wcschr()</b> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<b>wchar_t</b> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p><b>wcsrchr()</b> と <b>wsrchr()</b> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<b>wchar_t</b> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), windex()</b>	<p><b>windex()</b> と <b>windex()</b> の両関数の動作は、それぞれ <b>wcschr()</b> と <b>wcsrchr()</b> の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p><b>wcspbrk()</b> と <b>wcspbrk()</b> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wcstring(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspn()</b>	<p>wcsspn() と wsspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コードだけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspn(), wcscspn()</b>	<p>wcscspn() と wcscspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wcstring(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から 上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wcswcs(3C)

名前	wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wcswcs, wcssp, wssp, wcscsp, wscsp, wcstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wcschr(const wchar_t *ws, wchar_t wc); wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcswcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcssp(const wchar_t *ws1, const wchar_t *ws2); size_t wcscsp(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcscsp(const wchar_t *ws1, const wchar_t *ws2);</pre>

XPG4 と SUS  
デフォルトとその  
他の標準仕様



	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <code>ws</code>、<code>ws1</code>、および <code>ws2</code> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<code>ws2</code> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<code>ws1</code> が指すワイド文字列の終わりに追加します。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<code>ws2</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードを、<code>ws1</code> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<code>ws1</code> と <code>ws2</code> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<code>ws1</code> の指すワイド文字列中のコードが <code>ws2</code> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<code>ws1</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードと、<code>ws2</code> が指す配列の内容とを比較します。<code>ws1</code> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wcswcs(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p>wscpy() と wscpy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy() と wcsncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen() と wslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p>wcsrchr() と wcsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex() と wrindex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk() と wcspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される <code>wchar_t</code> ポインタを返します。<code>wchar_t</code> ポインタには、<code>wcstok()</code> 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は <code>wcstok()</code> の XPG4 と SUS のバージョンでは利用できません。また、<code>wstok()</code> 関数でも利用できません ( <code>standards(5)</code> 参照 )。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、<code>wcstok()</code> および <code>wstok()</code> は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、<code>wcstok()</code> と <code>wstok()</code> の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき <code>wcstok()</code> と <code>wstok()</code> の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wcswcs(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 wcswidth – ワイド文字列のカラム数

形式 #include <wchar.h>

```
int wcswidth(const wchar_t *pwcs, size_t n);
```

機能説明 wcswidth() 関数は、*pwcs* が示す文字列にあるワイド文字コード *n* (*n* 個のワイド文字を読み込む NULL ワイド文字コードがあった場合、ワイド文字コード *n* よりも少ない数) が必要とするカラム数を調べます。

戻り値 wcswidth() は、0 (*pwcs* が NULL ワイド文字コードを示す場合)、*pwcs* が示すワイド文字列に配置されたカラム数の位置、あるいは -1 (*pwcs* が示すワイド文字列にある最初のワイド文字コード *n* がワイド文字コードを印刷しない場合) を返します。

エラー 定義されたエラーはありません。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 setlocale(3C), wwidth(3C), attributes(5)

## wcsxfrm(3C)

名前 wcsxfrm, wsxfrm – ワイド文字列の変換

形式 #include <wchar.h>

```
size_t wcsxfrm(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

```
size_t wsxfrm(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

機能説明 wcsxfrm() と wsxfrm() の両関数は、ws2 が示すワイド文字列を変換して、その結果を ws1 が示す配列に書き出します。変換は、ある 2 つの変換済みワイド文字列に対して wcscmp(3C) または wscmp(3C) 関数を実行したとき、関数が返す値が変換前の文字列に対して wscoll(3C) または wcoll(3C) が返したであろう値 (0、0 より大きい、0 より小さいのいずれか) と一致する値を返すように行われます。n は、ws1 が指す配列に出力するワイド文字コードの最大数を表します。この数には、終端の NULL ワイド文字コードも含まれます。n が 0 の場合、ws1 は NULL ポインタでもかまいません。重なりあうオブジェクト間で複写を行うと、その結果は未定義です。

戻り値 wcsxfrm() および wsxfrm() 関数は、変換後のワイド文字列の長さを返します。これには終端の NULL ワイド文字コードは含まれません。戻り値が n 以上の値の場合、ws1 が示す配列の内容は予測できません。

エラーが発生すると、wcsxfrm() および wsxfrm() は ( size\_t ) -1 を返し、エラーを示す値を errno に設定します。

エラー wcsxfrm() および wsxfrm() 関数は、以下の状態のときエラーを返します。

EINVAL ws2 が示すワイド文字列に、照合手順に定義されていないワイド文字コードが含まれている。

ENOSYS 関数はサポートされていない。

使用法 変換処理は、変換後の任意の 2 つのワイド文字列を、wcscmp() または wscmp() がプログラムのロケール ( LC\_COLLATE カテゴリ ) 中の照合手順情報に従って正しく順序付けできるように行われます。

n が 0 であれば ws1 は NULL ポインタでもよい、という規則を利用して、変換前に ws1 の配列の大きさを調べることができます。

wcsxfrm() と wsxfrm() 関数は、setlocale(3C) がロケール変更で呼び出されない限り、マルチスレッドのアプリケーションでは安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

wcsxfrm(3C)

関連項目 | setlocale(3C), wscmp(3C), wscoll(3C), wscmp(3C), wscoll(3C),  
attributes(5)

## wctomb(3C)

名前 wctomb – ワイド文字コードを複数バイト文字に変換

形式 `#include <stdlib.h>`

```
int wctomb(char *s, wchar_t wchar);
```

機能説明 wctomb() 関数は、*wchar* の値を持つワイド文字コードに対応する文字を表すのに必要なバイト数を調べます。さらに、その文字表現 (複数バイト) を *s* が示す配列オブジェクト (*s* が NULL ポインタでないとき) に書き込みます。書き込まれる最大バイト数は MB\_CUR\_MAX が示す値です。

*s* に NULL ポインタを指定して呼び出すと、この関数は 0 を返します。この関数の動作は、現在のロケールの LC\_CTYPE カテゴリに左右されます。

戻り値 *s* が NULL ポインタの場合、wctomb() は、文字エンコーディングが状態依存型であれば 0 以外の値を、そうでなければ 0 を返します。*s* が NULL ポインタではない場合、*wchar* の値が正しい文字に対応していれば wctomb() はその文字を構成するバイト数を返し、正しい文字に対応していなければ -1 を返します。

どのような場合でも、返される値は MB\_CUR\_MAX マクロの値を超えることはありません。

エラー エラーは定義されていません。

使用法 マルチスレッドのアプリケーションにおいて、setlocale(3C) を呼び出してロケール変更を行っていない限り、wctomb() 関数は安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 mblen(3C), mbstowcs(3C), mbtowc(3C), setlocale(3C), wcstombs(3C), attributes(5)



名前 wctype – 文字クラスの定義

形式 #include <wchar.h>

```
wctype_t wctype(const char *charclass);
```

機能説明 wctype() 関数は、現ロケール中に定義されている文字クラス名用に定義されます。*charclass* 引数は、汎用文字クラスを表す文字列で、このクラスに対してはコードセット固有のタイプ情報が必要になります。以下の文字クラス名は、すべてのロケールに定義されています。

alnum	alpha	blank
cntrl	digit	graph
lower	print	punct
space	upper	xdigit

この他にも、ロケール定義ファイル (LC\_CTYPE カテゴリ) に定義されている文字クラス名が指定できます。

この関数は wctype\_t 型の値を返します。後続の iswctype(3C) 呼び出しの第 2 引数としてこの値を利用できます。wctype() が wctype\_t の値を決める際には、プログラムのロケール (LC\_CTYPE カテゴリ) 中の文字タイプ情報に定義されているコード化された文字セットの規則に従います。wctype() が返す値は、LC\_CTYPE カテゴリを変更する setlocale(3C) が呼び出されるまで有効です。

戻り値 指定された文字クラス名が現ロケール (LC\_CTYPE カテゴリ) に対して無効なとき、wctype() 関数は 0 を返します。有効であれば、wctype\_t 型のオブジェクトを返します。このオブジェクトは、iswctype() の呼び出しに使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 iswctype(3C), setlocale(3C), attributes(5)

## wcwidth(3C)

名前 | wcwidth – ワイド文字コードのカラム数

形式 | #include <wchar.h>  
int **wcwidth**(wchar\_t *wc*);

機能説明 | wcwidth() 関数は、ワイド文字 *wc* が必要とするカラム数を調べます。*wc* の値は、wchar\_t 型として表現可能な文字でなければならず、また現ロケール中の正しい文字に対応したワイド文字コードでなければなりません。

戻り値 | wcwidth() 関数は、0 (*wc* が NULL ワイド文字コードである場合)、ワイド文字コード *wc* が占有するカラム数の位置、あるいは -1 (*wc* が、印刷するワイド文字コードに対応していない場合) を返します。

エラー | 定義されたエラーはありません。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-Safe
CSI	対応済み

関連項目 | setlocale(3C), wcswidth(3C), attributes(5)

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsprk, wspbrk, wcswcs, wcssp, wssp, wcspsp, wscspn, wcstok, wstok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcssp(const wchar_t *ws1, const wchar_t *ws2);
size_t wcspsp(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchat_t wc);
wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wcspsp(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

## windex(3C)

	<pre> size_t <b>wcspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wcscat()</b> , <b>wscat()</b>	<p>wcscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscmp()</b> , <b>wscmp()</b>	<p>wcscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p>wcsncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p><i>wscpy()</i> と <i>wscpy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複製します。なお NULL のワイド文字コードに続くワイド文字コードは、複製の対象とはなりません。重なりあうオブジェクト間で複製が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wcslen()</b>	<p><i>wcslen()</i> と <i>wcslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p><i>wcschr()</i> と <i>wcschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p><i>wcsrchr()</i> と <i>wcsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), windex()</b>	<p><i>windex()</i> と <i>windex()</i> の両関数の動作は、それぞれ <i>wcschr()</i> と <i>wcsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p><i>wcspbrk()</i> と <i>wcspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

windex(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspn()</b>	<p>wcsspn() と wsspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspn(), wscspn()</b>	<p>wcscspn() と wscspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

### windex(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wrindex(3C)

名前	wcstring, wcsconcat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsprk, wspbrk, wcsvcs, wcsspn, wssp, wcsspn, wcsspn, wcsvcs, wcsvcs – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsvcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsvcs(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsvcs(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

### XPG4 と SUS

デフォルトとその  
他の標準仕様



	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <code>ws</code>、<code>ws1</code>、および <code>ws2</code> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<code>ws2</code> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<code>ws1</code> が指すワイド文字列の終わりに追加します。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<code>ws2</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードを、<code>ws1</code> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<code>ws1</code> と <code>ws2</code> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<code>ws1</code> の指すワイド文字列中のコードが <code>ws2</code> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<code>ws1</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードと、<code>ws2</code> が指す配列の内容とを比較します。<code>ws1</code> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wrindex(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p>wscpy() と wscpy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy() と wcsncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wcslen()</b>	<p>wcslen() と wcslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p>wcsrchr() と wcsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex() と wrindex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk() と wcspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される <code>wchar_t</code> ポインタを返します。<code>wchar_t</code> ポインタには、<code>wcstok()</code> 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は <code>wcstok()</code> の XPG4 と SUS のバージョンでは利用できません。また、<code>wstok()</code> 関数でも利用できません ( <code>standards(5)</code> 参照 )。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、<code>wcstok()</code> および <code>wstok()</code> は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、<code>wcstok()</code> と <code>wstok()</code> の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき <code>wcstok()</code> と <code>wstok()</code> の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wrindex(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 wstring, wscasecmp, wscasecmp, wsdup, wscol – ワイド文字の文字列操作

形式 #include <wchar.h>

```
int wscasecmp(const wchar_t *s1, const wchar_t *s2);
int wscasecmp(const wchar_t *s1, const wchar_t *s2, int n);
wchar_t *wsdup(const wchar_t *s);
int wscol(const wchar_t *s);
```

機能説明 これらの関数は、wchar\_t 型の NULL 文字で終わるワイド文字の文字列を操作します。これらのルーチンは、追加またはコピー中に、受け取る文字列のオーバーフロー状態を調べません。以下の説明の中で、s、s1 および s2 は、wchar\_t 型の NULL で終わるワイド文字の文字列を指します。

**wscasecmp(), wscasecmp()** wscasecmp() 関数は、引数を 大文字と小文字を区別せずに比較し、s1 が辞書順で s2 より大きいか、等しいかまたは小さいかによって、0 より大きいか、等しいかまたは小さい整数を返します。wscasecmp() も同じ比較を行います。4 種類の拡張 UNIX コード (EUC) は、コードセットの種類が異なる文字が比較される時、コードセット 0、2、3、1 と最下位から最高位の順に並べられます。

**wsdup()** wsdup() 関数は、s が指す文字列のコピーである新しいワイド文字列へのポインタを返します。新しい文字列用のメモリー領域は、malloc(3C) を用いて獲得します。新しい文字列を作成できない場合は、NULL ポインタが返されます。

**wscol()** wscol() 関数は、ワイド文字列 s の画面表示幅を カラム数で返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 malloc(3C), string(3C), wcstring(3C), attributes(5)

## wscat(3C)

名前	wcstring, wcsat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsprk, wspbrk, wcsvcs, wcsspn, wssp, wcsspn, wcsspn, wstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsvcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

### XPG4 と SUS

デフォルトとその  
他の標準仕様

	<pre> size_t wscspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *windex(const wchar_t *ws, wchar_t wc); wchar_t *wrindex(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *wcschr(const wchar_t *ws, wchar_t wc); const wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::wcschr(wchar_t *ws, wchar_t wc); wchar_t *std::wcpbrk(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <i>ws</i>、<i>ws1</i>、および <i>ws2</i> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<i>ws1</i> が指すワイド文字列の終わりに追加します。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<i>ws1</i> と <i>ws2</i> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<i>ws1</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードと、<i>ws2</i> が指す配列の内容とを比較します。<i>ws1</i> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

wscat(3C)

字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。

**wscpy(), wscopy()**

wscopy() と wscopy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。

**wscncpy(),  
wscncpy()**

wscncpy() と wscncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。

**wcslen(), wslen()**

wcslen() と wslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。

**wcschr(), wcschr()**

wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar\_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。

**wcsrchr(), wsrchr()  
( )**

wcsrchr() と wsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar\_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。

**windex(), wrindex()  
( )**

windex() と wrindex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。

**wcspbrk(),  
wspbrk()**

wcspbrk() と wspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。



<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wscat(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsprk, wspbrk, wcswcs, wcssp, wssp, wscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcssp(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchat_t wc);
wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

wchr(3C)

	<pre> size_t <b>wcspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p>wscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscmp()</b> , <b>wscmp()</b>	<p>wcscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p>wcsncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p><i>wscpy()</i> と <i>wscpy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複製します。なお NULL のワイド文字コードに続くワイド文字コードは、複製の対象とはなりません。重なりあうオブジェクト間で複製が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wcslen()</b>	<p><i>wcslen()</i> と <i>wcslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p><i>wcschr()</i> と <i>wcschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p><i>wcsrchr()</i> と <i>wcsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), windex()</b>	<p><i>windex()</i> と <i>windex()</i> の両関数の動作は、それぞれ <i>wcschr()</i> と <i>wcsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p><i>wcspbrk()</i> と <i>wcspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wschr(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspnl()</b>	<p>wcsspn() と wsspnl() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コードだけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspnl(), wscspnl()</b>	<p>wcscspnl() と wscspnl() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なってもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

wschr(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wscmp(3C)

名前	wcstring, wcsconcat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wcsspn, wcsspn, wstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

### XPG4 と SUS

デフォルトとその  
他の標準仕様



	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <code>ws</code>、<code>ws1</code>、および <code>ws2</code> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<code>ws2</code> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<code>ws1</code> が指すワイド文字列の終わりに追加します。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b> ( )	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<code>ws2</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードを、<code>ws1</code> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b> ( )	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<code>ws1</code> と <code>ws2</code> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<code>ws1</code> の指すワイド文字列中のコードが <code>ws2</code> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b> ( )	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<code>ws1</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードと、<code>ws2</code> が指す配列の内容とを比較します。<code>ws1</code> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wscmp(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p>wscpy() と wscpy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy() と wcsncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen() と wslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p>wcsrchr() と wcsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex() と wrindex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk() と wcspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される <code>wchar_t</code> ポインタを返します。<code>wchar_t</code> ポインタには、<code>wcstok()</code> 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は <code>wcstok()</code> の XPG4 と SUS のバージョンでは利用できません。また、<code>wstok()</code> 関数でも利用できません ( <code>standards(5)</code> 参照 )。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、<code>wcstok()</code> および <code>wstok()</code> は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、<code>wcstok()</code> と <code>wstok()</code> の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき <code>wcstok()</code> と <code>wstok()</code> の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wscmp(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 wstring, wcasecmp, wnccasecmp, wsdup, wscol – ワイド文字の文字列操作

形式 #include <wchar.h>

```
int wcasecmp(const wchar_t *s1, const wchar_t *s2);
int wnccasecmp(const wchar_t *s1, const wchar_t *s2, int n);
wchar_t *wsdup(const wchar_t *s);
int wscol(const wchar_t *s);
```

機能説明 これらの関数は、wchar\_t 型の NULL 文字で終わるワイド文字の文字列を操作します。これらのルーチンは、追加またはコピー中に、受け取る文字列のオーバーフロー状態を調べません。以下の説明の中で、s、s1 および s2 は、wchar\_t 型の NULL で終わるワイド文字の文字列を指します。

**wcasecmp(), wnccasecmp()** wcasecmp() 関数は、引数を 大文字と小文字を区別せずに比較し、s1 が辞書順で s2 より大きいか、等しいかまたは小さいかによって、0 より大きいか、等しいかまたは小さい整数を返します。wnccasecmp() も同じ比較を行います。比較するワイド文字の文字数は n 個以下です。4 種類の拡張 UNIX コード (EUC) は、コードセットの種類が異なる文字が比較されるとき、コードセット 0、2、3、1 と最下位から最高位の順に並べられます。

**wsdup()** wsdup() 関数は、s が指す文字列のコピーである新しいワイド文字列へのポインタを返します。新しい文字列用のメモリー領域は、malloc(3C) を用いて獲得します。新しい文字列を作成できない場合は、NULL ポインタが返されます。

**wscol()** wscol() 関数は、ワイド文字列 s の画面表示幅を カラム数で返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 malloc(3C), string(3C), wcstring(3C), attributes(5)

## wscoll(3C)

名前 wscoll, wscoll – 照合情報を使ってワイド文字列を比較

形式 #include <wchar.h>

```
int wcscoll(const wchar_t *ws1, const wchar_t *ws2);
```

```
int wscoll(const wchar_t *ws1, const wchar_t *ws2);
```

機能説明 `wcscoll()` と `wscoll()` の両関数は、`ws1` と `ws2` が示す 2 つのワイド文字列を比較します。両文字列は、現ロケールの `LC_COLLATE` カテゴリに従って解釈されます。

戻り値 比較処理が正常に終了すると、`wcscoll()` および `wscoll()` はその結果を示す値を返します。現ロケールに従って比較した結果、両文字列が一致していれば 0 を、`ws1` が指す文字列の方が `ws2` が指す文字列より大きければ 0 より大きい値を、小さければ 0 より小さい値をそれぞれ返します。エラーが発生した場合、`wcscoll()` および `wscoll()` は `errno` を設定することがありますが、エラー発生を表す戻り値は定義されていません。

エラー `wcscoll()` および `wscoll()` 関数は、以下の状態のときエラーを返します。

`EINVAL` 引数 `ws1` または `ws2` が示すワイド文字列に、照合手順に定義されていないワイド文字コードが含まれていた

`ENOSYS` 関数はサポートされていない

使用法 エラーを示す戻り値は定義されていないため、エラー発生の有無を知りたいアプリケーションは、`errno` を 0 に設定して `wcscoll()` または `wscoll()` を呼び出し、処理実行後に `errno` の値を調べる必要があります。値が 0 以外であれば、エラーが起こったものと判断できます。

大量のデータを分類するときは、`wcsxfrm(3C)` および `wcscmp(3C)` 関数を使用してください。

`wcscoll()` と `wscoll()` 関数は、`setlocale(3C)` を呼び出してロケールを変更しない限り、マルチスレッドのアプリケーションで安全に使用できます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

関連項目 `setlocale(3C)`, `wcscmp(3C)`, `wcsxfrm(3C)`, `attributes(5)`

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchat_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

## wscopy(3C)

	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p>wscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p>wscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p>wcsncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>



	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p><i>wscpy()</i> と <i>wscpy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複製します。なお NULL のワイド文字コードに続くワイド文字コードは、複製の対象とはなりません。重なりあうオブジェクト間で複製が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wcslen()</b>	<p><i>wcslen()</i> と <i>wcslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p><i>wcschr()</i> と <i>wcschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p><i>wcsrchr()</i> と <i>wcsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), windex()</b>	<p><i>windex()</i> と <i>windex()</i> の両関数の動作は、それぞれ <i>wcschr()</i> と <i>wcsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p><i>wcspbrk()</i> と <i>wcspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wscopy(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspn()</b>	<p>wcsspn() と wsspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspn(), wscspn()</b>	<p>wcscspn() と wscspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なってもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wscopy(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wcspn(3C)

名前	wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

XPG4 と SUS  
デフォルトとその  
他の標準仕様

	<pre> size_t wscspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *windex(const wchar_t *ws, wchar_t wc); wchar_t *wrindex(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *wcschr(const wchar_t *ws, wchar_t wc); const wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::wcschr(wchar_t *ws, wchar_t wc); wchar_t *std::wcpbrk(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <i>ws</i>、<i>ws1</i>、および <i>ws2</i> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<i>ws1</i> が指すワイド文字列の終わりに追加します。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<i>ws1</i> と <i>ws2</i> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<i>ws1</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードと、<i>ws2</i> が指す配列の内容とを比較します。<i>ws1</i> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wcspn(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p>wscpy() と wscpy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcncpy(), wcncpy()</b>	<p>wcncpy() と wcncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen() と wslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p>wcsrchr() と wsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex() と wrindex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk() と wcspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspн(), wsspн()</b>	wcsspн() と wsspн() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspн(), wscspн()</b>	wcscspн() と wscspн() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wscspn(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`



名前 wstring, wcasecmp, wnccasecmp, wsdup, wscol – ワイド文字の文字列操作

形式 #include <widec.h>

```
int wscasecmp(const wchar_t *s1, const wchar_t *s2);
int wncasecmp(const wchar_t *s1, const wchar_t *s2, int n);
wchar_t *wsdup(const wchar_t *s);
int wscol(const wchar_t *s);
```

機能説明 これらの関数は、wchar\_t 型の NULL 文字で終わるワイド文字の文字列を操作します。これらのルーチンは、追加またはコピー中に、受け取る文字列のオーバーフロー状態を調べません。以下の説明の中で、s、s1 および s2 は、wchar\_t 型の NULL で終わるワイド文字の文字列を指します。

**wscasecmp()**,  
**wncasecmp()** wscasecmp() 関数は、引数を 大文字と小文字を区別せずに比較し、s1 が辞書順で s2 より大きいか、等しいかまたは小さいかによって、0 より大きいか、等しいかまたは小さい整数を返します。wncasecmp() も同じ比較を行います。比較するワイド文字の文字数は n 個以下です。4 種類の拡張 UNIX コード (EUC) は、コードセットの種類が異なる文字が比較されるとき、コードセット 0、2、3、1 と最下位から最高位の順に並べられます。

**wsdup()** wsdup() 関数は、s が指す文字列のコピーである新しいワイド文字列へのポインタを返します。新しい文字列用のメモリー領域は、malloc(3C) を用いて獲得します。新しい文字列を作成できない場合は、NULL ポインタが返されます。

**wscol()** wscol() 関数は、ワイド文字列 s の画面表示幅を カラム数で返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 malloc(3C), string(3C), wcstring(3C), attributes(5)

## wslen(3C)

名前	wcstring, wcsat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wcsspn, wcsspn, wstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

XPG4 と SUS  
デフォルトとその  
他の標準仕様

	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <i>ws</i>、<i>ws1</i>、および <i>ws2</i> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<i>ws1</i> が指すワイド文字列の終わりに追加します。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<i>ws1</i> と <i>ws2</i> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<i>ws1</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードと、<i>ws2</i> が指す配列の内容とを比較します。<i>ws1</i> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

wslen(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p>wscpy() と wscpy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy() と wcsncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen() と wslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p>wcsrchr() と wcsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex() と wrindex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk() と wcspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wslen(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 wstring, wscasecmp, wscasecmp, wsdup, wscol – ワイド文字の文字列操作

形式 #include <wchar.h>

```
int wscasecmp(const wchar_t *s1, const wchar_t *s2);
int wscasecmp(const wchar_t *s1, const wchar_t *s2, int n);
wchar_t *wsdup(const wchar_t *s);
int wscol(const wchar_t *s);
```

機能説明 これらの関数は、wchar\_t 型の NULL 文字で終わるワイド文字の文字列を操作します。これらのルーチンは、追加またはコピー中に、受け取る文字列のオーバーフロー状態を調べません。以下の説明の中で、s、s1 および s2 は、wchar\_t 型の NULL で終わるワイド文字の文字列を指します。

**wscasecmp(), wscasecmp()** wscasecmp() 関数は、引数を 大文字と小文字を区別せずに比較し、s1 が辞書順で s2 より大きい、等しいかまたは小さいかによって、0 より大きい、等しいかまたは小さい整数を返します。wscasecmp() も同じ比較を行います。比較するワイド文字の文字数は n 個以下です。4 種類の拡張 UNIX コード (EUC) は、コードセットの種類が異なる文字が比較されるとき、コードセット 0、2、3、1 と最下位から最高位の順に並べられます。

**wsdup()** wsdup() 関数は、s が指す文字列のコピーである新しいワイド文字列へのポインタを返します。新しい文字列用のメモリー領域は、malloc(3C) を用いて獲得します。新しい文字列を作成できない場合は、NULL ポインタが返されます。

**wscol()** wscol() 関数は、ワイド文字列 s の画面表示幅を カラム数で返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 malloc(3C), string(3C), wcstring(3C), attributes(5)

## wsncat(3C)

名前	wcstring, wcsat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wcsncpy, wcsncpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsprk, wspbrk, wswcs, wcsspn, wssp, wcspspn, wcspspn, wstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wcschr(const wchar_t *ws, wchar_t wc); wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcspspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcspspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

### XPG4 と SUS

デフォルトとその  
他の標準仕様



	<pre> size_t wscspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *windex(const wchar_t *ws, wchar_t wc); wchar_t *wrindex(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *wcschr(const wchar_t *ws, wchar_t wc); const wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::wcschr(wchar_t *ws, wchar_t wc); wchar_t *std::wcpbrk(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <i>ws</i>、<i>ws1</i>、および <i>ws2</i> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<i>ws1</i> が指すワイド文字列の終わりに追加します。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscncat()</b> , <b>wsncat()</b>	<p><code>wscncat()</code> と <code>wsncat()</code> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<i>ws1</i> の終端の NULL ワイド文字コードは、<i>ws2</i> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <i>s1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<i>ws1</i> と <i>ws2</i> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wscncmp()</b> , <b>wsncmp()</b>	<p><code>wscncmp()</code> と <code>wsncmp()</code> の両関数は、<i>ws1</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードと、<i>ws2</i> が指す配列の内容とを比較します。<i>ws1</i> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wsncat(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscpy(), wscpy()</b>	<p>wscpy() と wscpy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy() と wcsncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen() と wslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p>wcsrchr() と wsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex() と wrindex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk() と wcspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	<b>wcswcs()</b> 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	<b>wcsspn()</b> と <b>wsspn()</b> の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	<b>wcscspn()</b> と <b>wscspn()</b> の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	<b>wcstok()</b> と <b>wstok()</b> を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される <code>wchar_t</code> ポインタを返します。<code>wchar_t</code> ポインタには、<b>wcstok()</b> 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は <b>wcstok()</b> の XPG4 と SUS のバージョンでは利用できません。また、<b>wstok()</b> 関数でも利用できません ( <a href="#">standards(5)</a> 参照 )。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、<b>wcstok()</b> および <b>wstok()</b> は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、<b>wcstok()</b> と <b>wstok()</b> の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき <b>wcstok()</b> と <b>wstok()</b> の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wsnecat(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wscspn, wscspn, wctok, wctok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchar_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wctok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wctok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <wchar.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchar_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

## wscmp(3C)

	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p>wscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p>wscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wscncmp()</b> , <b>wsncmp()</b>	<p>wscncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscopy(), wscopy()</b>	<p><i>wscopy()</i> と <i>wscopy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複写します。なお NULL のワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wcslen()</b>	<p><i>wcslen()</i> と <i>wcslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p><i>wcschr()</i> と <i>wcschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p><i>wcsrchr()</i> と <i>wcsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), windex()</b>	<p><i>windex()</i> と <i>windex()</i> の両関数の動作は、それぞれ <i>wcschr()</i> と <i>wcsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p><i>wcspbrk()</i> と <i>wcspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wsncmp(3C)

<b>wcswcs()</b>	<p><b>wcswcs()</b> 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspnl()</b>	<p><b>wcsspn()</b> と <b>wsspnl()</b> の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コードだけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspnl(), wscspnl()</b>	<p><b>wcscspnl()</b> と <b>wscspnl()</b> の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p><b>wcstok()</b> と <b>wstok()</b> を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される <code>wchar_t</code> ポインタを返します。<code>wchar_t</code> ポインタには、<b>wcstok()</b> 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は <b>wcstok()</b> の XPG4 と SUS のバージョンでは利用できません。また、<b>wstok()</b> 関数でも利用できません ( <a href="#">standards(5)</a> 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、<b>wcstok()</b> および <b>wstok()</b> は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、<b>wcstok()</b> と <b>wstok()</b> の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき <b>wcstok()</b> と <b>wstok()</b> の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>



## wsncmp(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wsncpy(3C)

名前	wcstring, wscat, wscat, wcsncat, wcsncat, wscmp, wscmp, wcsncmp, wcsncmp, wcsncpy, wcsncpy, wcsncpy, wcslen, wslen, wcschr, wcschr, wcsrchr, wsrchr, windex, wrindex, wcsprk, wsrprk, wswcs, wcsspn, wssp, wcsspn, wcsspn, wstok, wstok – ワイド文字列の操作
形式	<pre>#include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wcschr(const wchar_t *ws, wchar_t wc); wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);  #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);</pre>

XPG4 と SUS  
デフォルトとその  
他の標準仕様

	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <code>ws</code>、<code>ws1</code>、および <code>ws2</code> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<code>ws2</code> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<code>ws1</code> が指すワイド文字列の終わりに追加します。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<code>ws2</code> が指す配列中の <math>n</math> 個以下の数のワイド文字コードを、<code>ws1</code> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<code>ws1</code> と <code>ws2</code> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<code>ws1</code> の指すワイド文字列中のコードが <code>ws2</code> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<code>ws1</code> が指す配列中の <math>n</math> 個以下の数のワイド文字コードと、<code>ws2</code> が指す配列の内容とを比較します。<code>ws1</code> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

## wscncpy(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscsncpy(), wscsncpy()</b>	<p>wscsncpy() と wscsncpy() の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy() と wcsncpy() の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wcslen()</b>	<p>wcslen() と wcslen() の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr() と wcschr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p>wcsrchr() と wcsrchr() の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), windex()</b>	<p>windex() と windex() の両関数の動作は、それぞれwcschr() と wcsrchr() の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk() と wcspbrk() の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wsncpy(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsbrk, wspbrk, wcs wcs, wcsspn, wssp, wscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcs wcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchat_t wc);
wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

## wspbrk(3C)

	<pre> size_t <b>wcspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wcscat()</b> , <b>wscat()</b>	<p>wcscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscmp()</b> , <b>wscmp()</b>	<p>wcscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p>wcsncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>



	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscopy(), wscopy()</b>	<p><i>wscopy()</i> と <i>wscopy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複写します。なお NULL のワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wcslen()</b>	<p><i>wcslen()</i> と <i>wcslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p><i>wcschr()</i> と <i>wcschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p><i>wcsrchr()</i> と <i>wcsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), windex()</b>	<p><i>windex()</i> と <i>windex()</i> の両関数の動作は、それぞれ <i>wcschr()</i> と <i>wcsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p><i>wcspbrk()</i> と <i>wcspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wspbrk(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspn()</b>	<p>wcsspn() と wsspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspn(), wscspn()</b>	<p>wcscspn() と wscspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

### wspbrk(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wsprintf(3C)

名前 | wsprintf – 書式付き出力変換

形式 | #include <stdio.h>  
#include <wchar.h>

```
int wsprintf(wchar_t *s, const char *format, /* arg */ ... );
```

機能説明 | wsprintf() 関数は、ワイド文字 (wchar\_t) 型の NULL 文字で終わるワイド文字列を出力します。ユーザーは、この wchar\_t 文字列を格納できる十分な空間を割り当てる必要があります。

wsprintf() は、書き込まれたワイド文字の (NULL 終了文字を除く) 文字数を返します。wsprintf() の変換指定および実行動作は、通常の sprintf(3C) 関数と同じです。ただし、wsprintf() の場合は、ワイド文字列が出力され、sprintf() の場合は拡張 UNIX コード (EUC) 文字列が出力されます。

戻り値 | 正常終了の場合は、出力される文字数を返します。エラーの場合は、負の値を返します。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 | wsscanf(3C), printf(3C), scanf(3C), sprintf(3C), attributes(5)

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchat_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchat_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

wsrchr(3C)

	<pre> size_t <b>wcspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wcscat()</b> , <b>wscat()</b>	<p>wcscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscmp()</b> , <b>wscmp()</b>	<p>wcscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p>wcsncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は 0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wcscpy(), wscpy()</b>	<p><i>wcscpy()</i> と <i>wscpy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複製します。なお NULL のワイド文字コードに続くワイド文字コードは、複製の対象とはなりません。重なりあうオブジェクト間で複製が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p><i>wcslen()</i> と <i>wslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wschr()</b>	<p><i>wcschr()</i> と <i>wschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p><i>wcsrchr()</i> と <i>wsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), wrindex()</b>	<p><i>windex()</i> と <i>wrindex()</i> の両関数の動作は、それぞれ <i>wschr()</i> と <i>wsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wspbrk()</b>	<p><i>wcspbrk()</i> と <i>wspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wsrchr(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspnl()</b>	<p>wcsspn() と wsspnl() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspnl(), wscspnl()</b>	<p>wcscspnl() と wscspnl() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>



## wsrchr(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wsscanf(3C)

名前 wsscanf – 書式付き入力変換

形式 

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

```
int wsscanf(wchar_t *s, const char *format, /* pointer */ ... );
```

機能説明 wsscanf() 関数は、ワイド文字列 *s* からワイド文字を読み取り、書式にしたがってそれを解釈し、その結果を引数に格納します。wsscanf() には、引数として制御文字列 *format* および変換した入力の格納場所を示す一連の *pointer* を指定して下さい。書式に対する十分な引数が指定されない場合は、結果は不定となります。書式が終了しても引数が残っている場合は、残りの引数は無視されます。

wsscanf() の変換指定と実行動作は、通常の sscanf(3C) と同じです。ただし、wsscanf() の場合はワイド文字列から文字を読み取り、sscanf() の場合は拡張 UNIX コード (EUC) 文字列から文字を読み取ります。

戻り値 wsscanf() は、正常終了の場合、変換対象となった文字数を返します。エラーの場合は、負の値を返します。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 wprintf(3C), printf(3C), scanf(3C), attributes(5)

名前 | wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcpbrk, wcpbrk, wswcs, wcsspn, wssp, wscspn, wscspn, wctok, wtok – ワイド文字列の操作

形式 | #include <wchar.h>

```

wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wcschr(const wchar_t *ws, wchar_t wc);
wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wswcs(const wchar_t *ws1, const wchar_t *ws2);
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wctok(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wctok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
#include <widec.h>
wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
int wscmp(const wchar_t *ws1, const wchar_t *ws2);
int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
size_t wslen(const wchar_t *ws);
wchar_t *wschr(const wchar_t *ws, wchar_t wc);
wchar_t *wsrchr(const wchar_t *ws, wchar_t wc);
wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);
size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

```

XPG4 と SUS  
デフォルトとその  
他の標準仕様

## wsspn(3C)

	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;cwchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、wchar_t 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では ws、ws1、および ws2 は、wchar_t 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p>wscat() と wscat() の両関数は、ws2 が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、ws1 が指すワイド文字列の終わりに追加します。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p>wcsncat() と wsncat() の両関数は、ws2 が指す配列中の n 個以下の数のワイド文字コードを、ws1 が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。ws1 の終端の NULL ワイド文字コードは、ws2 の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに s1 を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p>wscmp() と wscmp() の両関数は、ws1 と ws2 が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。ws1 の指すワイド文字列中のコードが ws2 の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p>wcsncmp() と wsncmp() の両関数は、ws1 が指す配列中の n 個以下の数のワイド文字コードと、ws2 が指す配列の内容とを比較します。ws1 の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。 <i>ws1</i> の指すワイド文字列中のコードが <i>ws2</i> の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wcscpy(), wscpy()</b>	<p><i>wcscpy()</i> と <i>wscpy()</i> の両関数は、<i>ws2</i> が指すワイド文字列 (終端の NULL ワイド文字コードも含む) を、<i>ws1</i> が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p><i>wcsncpy()</i> と <i>wcsncpy()</i> の両関数は、<i>ws2</i> が指す配列中の <i>n</i> 個以下の数のワイド文字コードを、<i>ws1</i> が指す配列に複製します。なお NULL のワイド文字コードに続くワイド文字コードは、複製の対象とはなりません。重なりあうオブジェクト間で複製が行われた場合、その結果は未定義です。<i>ws2</i> が指す配列が、<i>n</i> 個未満のワイド文字コードからなるワイド文字列の場合、<i>ws1</i> が指す出力先の配列には、合計コード数が <i>n</i> 個に達するまで NULL ワイド文字コードが書き込まれます。両関数ともに <i>ws1</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p><i>wcslen()</i> と <i>wslen()</i> の両関数は、<i>ws</i> が指すワイド文字列中のワイド文字コードの個数を計算します。終端の NULL ワイド文字コードは、数には含まれません。両関数ともに <i>ws</i> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p><i>wcschr()</i> と <i>wcschr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最初の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>wcsrchr(), wcsrchr()</b>	<p><i>wcsrchr()</i> と <i>wcsrchr()</i> の両関数は、<i>ws</i> が指すワイド文字列内にワイド文字コード <i>wc</i> が含まれていれば、最後の出現位置を報告します。<i>wc</i> の値は、<i>wchar_t</i> 型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端の NULL ワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。</p>
<b>windex(), windex()</b>	<p><i>windex()</i> と <i>windex()</i> の両関数の動作は、それぞれ <i>wcschr()</i> と <i>wcsrchr()</i> の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p><i>wcspbrk()</i> と <i>wcspbrk()</i> の両関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければ NULL ポインタを返します。</p>

## wsspn(3C)

<b>wcswcs()</b>	<p>wcswcs() 関数は、<i>ws1</i> が指すワイド文字列内に、<i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、<i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。<i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。</p>
<b>wcsspn(), wsspn()</b>	<p>wcsspn() と wsspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcscspn(), wscspn()</b>	<p>wcscspn() と wscspn() の両関数は、<i>ws1</i> が指すワイド文字列を先頭から検査し、<i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcstok(), wstok()</b>	<p>wcstok() と wstok() を連続して呼び出すと、<i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、<i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。</p>
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

wsspn(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

## wstod(3C)

名前	wcstod, wstod, watof – ワイド文字列を倍精度浮動小数点数に変換
形式	<pre>#include &lt;wchar.h&gt;  double wcstod(const wchar_t *nptr, wchar_t **endptr); double wstod(const wchar_t *nptr, wchar_t **endptr); double watof(wchar_t *nptr);</pre>
機能説明	<p>wcstod() と wstod() の両関数は、<i>nptr</i> が示すワイド文字列の先頭部分を double 型の表現に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。初めの部分は、iswspace(3C) で識別される空白のワイド文字コードの並びで、空の場合もあります。次が変換対象となる文字コードの並びで、浮動小数点数として解釈されます。最後の部分は、解釈不能なワイド文字コードの並びで、これには入力文字列の終わりの NULL ワイド文字コードも含まれます。分割後、変換対象となる文字コードの並びを浮動小数点数に変換しようと試み、その結果を返します。</p> <p>変換対象となる文字コードの並びとして期待される形式は、先頭が + または - の符号 (なくても可) で、その後が小数点を含むまたは含まない 1 つ以上の数字の集まりで、必要であれば最後に指数部が続きます。指数部は、文字 e または E で、その後にはオプションとして符号やいくつかの 10 進数を付加することもできます。変換対象となる文字コードの並びとは、入力ワイド文字列中の最初の空白でないワイド文字コードで始まる、期待された形式の、最長のワイド文字コードの並びであると定義できます。入力ワイド文字列が空の場合、または空白を表すワイド文字コードだけからなる場合、または最初の空白でないワイド文字コードが符号でも数字でも小数点でもない場合、変換対象となる文字コードの並びにはワイド文字コードが 1 つも含まれません。</p> <p>変換対象となる文字コードの並びの形式が期待どおりの場合、最初の数字または小数点 (どちらか最初に現れた方) で始まる一連のワイド文字コードは、C 言語で定義される浮動小数点数であると解釈されます。C 言語の定義と異なる点は、ピリオドの代わりに小数点が使われること、指数も小数点も現れなければワイド文字列中の最後の数字の後に小数点が続くと見なされることです。変換対象となる文字コードの並びが負の符号 (-) で始まっている場合、変換の結果として得られた値は負となります。<i>endptr</i> が示すオブジェクトには、最終部分のワイド文字列へのポインタが書き込まれます。ただし <i>endptr</i> が NULL ポインタの場合を除きます。</p> <p>小数点はプログラムのロケール (LC_NUMERIC カテゴリ) 中に定義されます。POSIX ロケール、および小数点が定義されていないロケールにおいては、小数点のデフォルト値はピリオド (.) となります。</p> <p>ロケールが POSIX 以外の場合、そのシステム固有の他の変換対象となる文字コードの並び形式も使用できます。</p> <p>変換対象となる文字コードの並びが空の場合、または期待された形式ではない場合、変換処理は行われません。またこのとき、<i>endptr</i> が示すオブジェクトには、<i>nptr</i> の値が書き込まれます。ただし <i>endptr</i> が NULL ポインタの場合を除きます。</p> <p>watof(str) 関数は wstod(str, (wchar_t **)NULL) と同じです。</p>



戻り値 `wcstod()` および `wstod()` 関数は、変換された値 (もしあれば) を返します。変換が行われなかった場合、0 が返されます。このとき、`errno` は `EINVAL` の値に設定されることがあります。

正しい値が表現可能な値の範囲外であれば、`±HUGE_VAL` のどれか (値の符号による) が返され、`errno` は `ERANGE` の値に設定されます。

正しい値がアンダフローを起こしてしまうのであれば、0 が返され、`errno` は `ERANGE` の値に設定されます。

エラー `wcstod()` および `wstod()` 関数は、以下の状態のときエラーを返します。

`ERANGE` 返されるべき値が、オーバフローまたはアンダフローを起こしてしまう

`wcstod()` および `wcstod()` 関数は、以下の状態のときエラーを返します。

`EINVAL` 変換が実行できない

使用法 エラー発生時には 0 が返されますが、これは正常終了の場合にも返される値です。したがってエラー発生の有無を知りたいアプリケーションは、`errno` を 0 に設定して `wcstod()` または `wstod()` を呼び出し、処理実行後に `errno` の値を調べる必要があります。値が 0 以外であれば、エラーが起こったものと判断できます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 `iswspace(3C)`, `localeconv(3C)`, `scanf(3C)`, `setlocale(3C)`, `wcstol(3C)`, `attributes(5)`

## wstok(3C)

名前	wcstring, wcsat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsprk, wspbrk, wcsvcs, wcsspn, wssp, wcsspn, wscspn, wscspn, wstok, wstok – ワイド文字列の操作
形式	<pre> #include &lt;wchar.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wcslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wcsvcs(const wchar_t *ws1, const wchar_t *ws2); size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2); size_t wscspn(const wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2); wchar_t *wstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr); #include &lt;widec.h&gt;  wchar_t *wscat(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncat(wchar_t *ws1, const wchar_t *ws2, size_t n); int wscmp(const wchar_t *ws1, const wchar_t *ws2); int wsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n); wchar_t *wscpy(wchar_t *ws1, const wchar_t *ws2); wchar_t *wsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n); size_t wslen(const wchar_t *ws);  wchar_t *wschr(const wchar_t *ws, wchar_t wc); wchar_t *wsrchr(const wchar_t *ws, wchar_t wc); wchar_t *wspbrk(const wchar_t *ws1, const wchar_t *ws2); size_t wscspn(const wchar_t *ws1, const wchar_t *ws2); </pre>

## XPG4 と SUS

デフォルトとその  
他の標準仕様

	<pre> size_t <b>wscspn</b>(const wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>wstok</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *<b>windex</b>(const wchar_t *ws, wchar_t wc); wchar_t *<b>wrindex</b>(const wchar_t *ws, wchar_t wc); </pre>
ISO C++	<pre> #include &lt;wchar.h&gt;  const wchar_t *<b>wcschr</b>(const wchar_t *ws, wchar_t wc); const wchar_t *<b>wcspbrk</b>(const wchar_t *ws1, const wchar_t *ws2); const wchar_t *<b>wcsrchr</b>(const wchar_t *ws, wchar_t wc);  #include &lt;wchar&gt;  wchar_t *std::<b>wcschr</b>(wchar_t *ws, wchar_t wc); wchar_t *std::<b>wcspbrk</b>(wchar_t *ws1, const wchar_t *ws2); wchar_t *std::<b>wcsrchr</b>(wchar_t *ws, wchar_t wc); </pre>
機能説明	<p>これらの関数は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を扱います。追加またはコピー処理を行うとき、受け取り側の文字列でオーバーフロー状態が発生してもこれらの関数はそれをチェックしません。以下の説明では <code>ws</code>、<code>ws1</code>、および <code>ws2</code> は、<code>wchar_t</code> 型の NULL 文字で終了するワイド文字列を指しています。</p>
<b>wscat()</b> , <b>wscat()</b>	<p><code>wscat()</code> と <code>wscat()</code> の両関数は、<code>ws2</code> が指すワイド文字列 ( 終端の NULL ワイド文字コードも含む ) をコピーして、<code>ws1</code> が指すワイド文字列の終わりに追加します。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncat()</b> , <b>wsncat()</b>	<p><code>wcsncat()</code> と <code>wsncat()</code> の両関数は、<code>ws2</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードを、<code>ws1</code> が指すワイド文字列の終わりに追加します。なお NULL のワイド文字コードとそれに続くワイド文字コードは、追加の対象とはなりません。<code>ws1</code> の終端の NULL ワイド文字コードは、<code>ws2</code> の先頭のワイド文字コードにより置き換えられます。追加されるワイド文字コードの終わりには、必ず NULL ワイド文字コードが付加されます。両関数ともに <code>s1</code> を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wscmp()</b> , <b>wscmp()</b>	<p><code>wscmp()</code> と <code>wscmp()</code> の両関数は、<code>ws1</code> と <code>ws2</code> が示す 2 つのワイド文字列を比較します。両ワイド文字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0 以外の値を返します。<code>ws1</code> の指すワイド文字列中のコードが <code>ws2</code> の指すワイド文字列中のコードより大きい場合は 0 より大きい値を返し、逆に小さい場合は 0 より小さい値を返します。両ワイド文字列が一致する場合には 0 を返します。</p>
<b>wcsncmp()</b> , <b>wsncmp()</b>	<p><code>wcsncmp()</code> と <code>wsncmp()</code> の両関数は、<code>ws1</code> が指す配列中の <code>n</code> 個以下の数のワイド文字コードと、<code>ws2</code> が指す配列の内容とを比較します。<code>ws1</code> の配列の中で、NULL ワイド文字コード以降のワイド文字コードは比較の対象とはなりません。両ワイド文</p>

wstok(3C)

	<p>字列中のコードを順番に比較して、両ワイド文字列が一致しない場合は、最初に一致しない両ワイド文字列の違いに応じて、0以外の値を返します。ws1の指すワイド文字列中のコードがws2の指すワイド文字列中のコードより大きい場合は0より大きい値を返し、逆に小さい場合は0より小さい値を返します。両ワイド文字列が一致する場合には0を返します。</p>
<b>wscspy(), wscopy()</b>	<p>wscspy()とwscopy()の両関数は、ws2が指すワイド文字列(終端のNULLワイド文字コードも含む)を、ws1が指す配列にコピーします。重なりあうオブジェクト間でコピーが行われた場合、その結果は未定義です。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcsncpy(), wcsncpy()</b>	<p>wcsncpy()とwcsncpy()の両関数は、ws2が指す配列中のn個以下の数のワイド文字コードを、ws1が指す配列に複写します。なおNULLのワイド文字コードに続くワイド文字コードは、複写の対象とはなりません。重なりあうオブジェクト間で複写が行われた場合、その結果は未定義です。ws2が指す配列が、n個未満のワイド文字コードからなるワイド文字列の場合、ws1が指す出力先の配列には、合計コード数がn個に達するまでNULLワイド文字コードが書き込まれます。両関数ともにws1を返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcslen(), wslen()</b>	<p>wcslen()とwslen()の両関数は、wsが指すワイド文字列中のワイド文字コードの個数を計算します。終端のNULLワイド文字コードは、数には含まれません。両関数ともにwsを返します。エラー発生を示す戻り値は定義されていません。</p>
<b>wcschr(), wcschr()</b>	<p>wcschr()とwcschr()の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最初の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最初の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>wcsrchr(), wsrchr()</b>	<p>wcsrchr()とwsrchr()の両関数は、wsが指すワイド文字列内にワイド文字コードwcが含まれていれば、最後の出現位置を報告します。wcの値は、wchar_t型として表示可能な文字である必要があり、また現ロケール中の正しい文字に対応したワイド文字コードである必要があります。終端のNULLワイド文字コードは、ワイド文字列の一部と見なされます。処理が正常に終了すると、ワイド文字コードが見つければその最後の出現位置を示すポインタを返し、見つからなければNULLポインタを返します。</p>
<b>windex(), wrindex()</b>	<p>windex()とwrindex()の両関数の動作は、それぞれwcschr()とwsrchr()の動作と同じです。</p>
<b>wcspbrk(), wcspbrk()</b>	<p>wcspbrk()とwcspbrk()の両関数は、ws1が指すワイド文字列内に、ws2が指すワイド文字列中のいずれかのワイド文字コードが含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、ws2中のいずれかのワイド文字コードが見つければその最初の出現位置を示すポインタを返し、1つも見つからなければNULLポインタを返します。</p>

<b>wcswcs()</b>	wcswcs() 関数は、 <i>ws1</i> が指すワイド文字列内に、 <i>ws2</i> が指すワイド文字列 ( 終端の NULL ワイド文字コードを除く ) が含まれていれば、その最初の出現位置を報告します。処理が正常に終了すると、 <i>ws2</i> 中のワイド文字列が見つければその最初の出現位置を示すポインタを返し、見つからなければ NULL ポインタを返します。 <i>ws2</i> が長さ 0 のワイド文字列を指している場合、関数は <i>ws1</i> を返します。
<b>wcsspn(), wsspn()</b>	wcsspn() と wsspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード だけで構成される部分の最大長を計算します。両関数ともに最初の部分文字列 <i>ws1</i> の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcscspn(), wscspn()</b>	wcscspn() と wscspn() の両関数は、 <i>ws1</i> が指すワイド文字列を先頭から検査し、 <i>ws2</i> が示すワイド文字列中のワイド文字コード 以外のコードだけで構成される最初の部分の最大長を計算します。両関数ともに <i>ws1</i> の初期部分文字列の長さを返します。エラー発生を示す戻り値は定義されていません。
<b>wcstok(), wstok()</b>	wcstok() と wstok() を連続して呼び出すと、 <i>ws1</i> が指すワイド文字列はいくつかのトークンに分割されます。トークンは、 <i>ws2</i> が指すワイド文字列から得られるワイド文字コードによって区切られます。
デフォルトとその他の標準仕様	<p>第 3 引数は、呼び出し元から提供される wchar_t ポインタを返します。wchar_t ポインタには、wcstok() 関数が同じワイド文字列を続けて走査するために、必要な情報が格納されています。この引数は wcstok() の XPG4 と SUS のバージョンでは利用できません。また、wstok() 関数でも利用できません ( standards(5) 参照)。</p> <p>最初の呼び出しでは <i>ws1</i> を第 1 引数に指定し、それ以後の呼び出しでは NULL ポインタを第 1 引数に指定します。トークンの区切り文字として指定する文字列、つまり <i>ws2</i> が指すワイド文字列は、呼び出しごとに異なっていてもかまいません。</p> <p>1 回目の呼び出しでは、<i>ws1</i> が示すワイド文字列を先頭から検査して、<i>ws2</i> が指す区切り文字列に含まれていないワイド文字コードを探します。そのようなワイド文字コードが最初に見つかった位置が、第 1 トークンの始まりとなります。そのようなワイド文字コードが 1 つも見つからなければ、<i>ws1</i> が指すワイド文字列にはトークンは存在しないことになり、wcstok() および wstok() は NULL ポインタを返します。</p> <p>第 1 トークンの始まりが定義されたら、wcstok() と wstok() の両関数は、その地点からさらに検査して、区切り文字列中に含まれているワイド文字コードを探します。<i>ws1</i> の終わりまで検索してもそのようなワイド文字コードが見つからなければ、文字列全体の終わりがトークンの終わりとなり、以降の呼び出しに対しては、NULL ポインタが返されます。ワイド文字コードが見つければ、その地点が第 1 トークンの終端となり、そのコードは NULL ワイド文字に置き換えられます。このとき wcstok() と wstok() の両関数は、その次のワイド文字コードへのポインタを保存します。次のトークン検索は、そのポインタの位置から開始されます。</p>

## wstok(3C)

このようにして、第 1 引数に NULL ポインタを指定して `wcstok()` または `wstok()` を連続して呼び出すと、保存されたポインタの位置から上記のようなトークンの検索が実行されます。処理が正常に終了すると、両関数は生成したトークンの先頭のワイド文字コードを指すポインタを返します。正常に終了しなかった場合、NULL ポインタが返されます。

属性 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-safe
CSI	対応済み

関連項目 `malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

名前 wcstol, wstol, watol, watoll, watoi – ワイド文字列を整数に変換

```
形式 #include <wchar.h>

long int wcstol(const wchar_t *nptr, wchar_t **endptr, int base);

#include <widec.h>

long int wstol(const wchar_t *nptr, wchar_t **endptr, int base);

long watol(wchar_t *nptr);

long long watoll(wchar_t *nptr);

int watoi(wchar_t *nptr);
```

機能説明

`wcstol()` と `wstol()` の両関数は、`nptr` が示すワイド文字列の先頭部分を `long int` 型の整数表現に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。初めの部分は、`iswspace(3C)` で識別される空白のワイド文字コードの並びで、空の場合もあります。次が変換対象となる文字コードの並びで、`base` の値から決定される指数表現の整数として解釈されます。最後の部分は、解釈不能なワイド文字コードの並びで、これには入力文字列の終わりの `NULL` ワイド文字コードも含まれます。分割後、変換対象となる文字コードの並びを整数に変換しようと試み、その結果を返します。

`base` の値が 0 のとき、変換対象となる文字コードの並びの値は 10 進、8 進、または 16 進の定数であるものと期待されます。定数には + または - の符号が付いていてもかまいません。10 進定数は 0 以外の数字で始まり、いくつかの 10 進数で構成されます。8 進定数は接頭辞 0 で認識され、その後には 0 から 7 までの数字をいくつか続けることができます。16 進定数は、接頭辞 0x または 0X で認識され、その後には 0 から 9 までの数字および a (または A) から f (または F) までの文字がいくつか続きます。

`base` の値が 2 から 36 までの範囲のとき、変換対象となる文字コードの並びの形式は `base` が示す基数を用いた整数を表す一連の数字または文字であると期待されます。先頭に + または - の符号が付加されることはありませんが、整数の接尾辞は含まれません。文字 a (または A) から z (または Z) は、それぞれ 10 から 35 までの値に対応します。`base` の値以内の範囲の文字だけが、数値に対応することができます。`base` の値が 16 のとき、値を表す一連の数字や文字の前に、0x または 0X のワイド文字コード表現を付加することができます。符号が存在していれば、その後となります。

変換対象となる文字コードの並びとは、入力ワイド文字列中の最初の空白以外のワイド文字コードで始まる、期待された形式の、最長のワイド文字コードの並びであると定義できます。入力ワイド文字列が空の場合、または空白を表すワイド文字コードだけからなる場合、または最初の空白以外のワイド文字コードが符号でもなく指定可能な数字や文字でもない場合、変換対象となる文字コードの並びにはワイド文字コードが 1 つも含まれません。

変換対象となる文字コードの並びの形式が期待どおりのものであり、`base` の値が 0 の場合、最初の数字で始まる一連のワイド文字コードは、整定数であると解釈されます。変換対象となる文字コードの並びの形式が期待どおりのものであり、`base` の値が 2 から 36 までの範囲にある場合、その値を基数として変換が行われます。文字に対しては、前述したような方法で数値が割り当てられます。変換対象となる文字コード

## wstol(3C)

の並びが負の符号 (-) で始まっている場合、変換の結果として得られる値は負となります。 *endptr* が示すオブジェクトには、最終部分のワイド文字列へのポインタが書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

ロケールが POSIX 以外の場合、そのシステム固有の他の変換対象となる文字コードの並び形式も使用できます。

変換対象となる文字コードの並びが空の場合、または期待された形式ではない場合、変換処理は行われません。またこのとき、 *endptr* が示すオブジェクトには、 *nptr* の値が書き込まれます。ただし *endptr* が NULL ポインタの場合を除きます。

`watol()` 関数は `wstol(str, (wchar_t **)NULL, 10)` と同機能です。

`watoll()` 関数は `longlong` 型の `watol()` です。

`watoi()` 関数は `(int)watol()` と同機能です。

**戻り値** 処理が正常に終了すると、 `wcstol()` および `wstol()` は、変換された値(もしあれば)を返します。変換が行われなかった場合、0 が返されます。このとき、 `errno` はエラー状態を示す値に設定されることがあります。正しい値が表現可能な値の範囲外であれば、 `{LONG_MAX}` または `{LONG_MIN}` のどちらか(値の符号による)が返され、 `errno` は `ERANGE` の値に設定されます。

**エラー** `wcstol()` および `wstol()` 関数は、以下の条件のとき異常終了します。

`EINVAL` *base* の値はサポートされていない

`ERANGE` 返されるべき値は表現不能である

また `wcstol()` および `wstol()` 関数は、以下の状態のときエラーを返します。

`EINVAL` 変換が実行できない

**属性** 次の属性については `attributes(5)` のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

**関連項目** `iswalph(3C)`, `iswspace(3C)`, `scanf(3C)`, `wcstod(3C)`, `attributes(5)`

**注意事項** エラー発生時には 0、 `{LONG_MIN}`、および `{LONG_MAX}` が返されますが、これらは正常終了の場合にも返される値です。したがってエラー発生の有無を知りたいアプリケーションは、 `errno` を 0 に設定して `wcstol()` または `wstol()` を呼び出し、処理実行後に `errno` の値を調べる必要があります。値が 0 以外であれば、エラーが起こったものと判断できます。

代入時に必要であれば、または明示的に指定されていれば、 `longlong` 型から `long` 型への切り捨てが発生することがあります。



名前 | wstring, wscasecmp, wsnccasecmp, wsdup, wscol – ワイド文字の文字列操作

形式 | #include <wchar.h>

```
int wscasecmp(const wchar_t *s1, const wchar_t *s2);
int wsnccasecmp(const wchar_t *s1, const wchar_t *s2, int n);
wchar_t *wsdup(const wchar_t *s);
int wscol(const wchar_t *s);
```

機能説明 | これらの関数は、wchar\_t 型の NULL 文字で終わるワイド文字の文字列を操作します。これらのルーチンは、追加またはコピー中に、受け取る文字列のオーバーフロー状態を調べません。以下の説明の中で、s、s1 および s2 は、wchar\_t 型の NULL で終わるワイド文字の文字列を指します。

**wscasecmp()**,  
**wsnccasecmp()** | wscasecmp() 関数は、引数を 大文字と小文字を区別せずに比較し、s1 が辞書順で s2 より大きいか、等しいかまたは小さいかによって、0 より大きいか、等しいかまたは小さい整数を返します。wsnccasecmp() も同じ比較を行います。比較するワイド文字の文字数は n 個以下です。4 種類の拡張 UNIX コード (EUC) は、コードセットの種類が異なる文字が比較される時、コードセット 0、2、3、1 と最下位から最高位の順に並べられます。

**wsdup()** | wsdup() 関数は、s が指す文字列のコピーである新しいワイド文字列へのポインタを返します。新しい文字列用のメモリー領域は、malloc(3C) を用いて獲得します。新しい文字列を作成できない場合は、NULL ポインタが返されます。

**wscol()** | wscol() 関数は、ワイド文字列 s の画面表示幅を カラム数で返します。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 | malloc(3C), string(3C), wcstring(3C), attributes(5)

## wsxfrm(3C)

名前 wcsxfrm, wsxfrm – ワイド文字列の変換

形式 #include <wchar.h>

```
size_t wcsxfrm(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

```
size_t wsxfrm(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

機能説明 wcsxfrm() と wsxfrm() の両関数は、ws2 が示すワイド文字列を変換して、その結果を ws1 が示す配列に書き出します。変換は、ある 2 つの変換済みワイド文字列に対して wcscmp(3C) または wscmp(3C) 関数を実行したとき、関数が返す値が変換前の文字列に対して wcscoll(3C) または wscoll(3C) が返したであろう値 (0、0 より大きい、0 より小さいのいずれか) と一致する値を返すように行われます。n は、ws1 が指す配列に出力するワイド文字コードの最大数を表します。この数には、終端の NULL ワイド文字コードも含まれます。n が 0 の場合、ws1 は NULL ポインタでもかまいません。重なりあうオブジェクト間で複写を行うと、その結果は未定義です。

戻り値 wcsxfrm() および wsxfrm() 関数は、変換後のワイド文字列の長さを返します。これには終端の NULL ワイド文字コードは含まれません。戻り値が n 以上の値の場合、ws1 が示す配列の内容は予測できません。

エラーが発生すると、wcsxfrm() および wsxfrm() は ( size\_t ) -1 を返し、エラーを示す値を errno に設定します。

エラー wcsxfrm() および wsxfrm() 関数は、以下の状態のときエラーを返します。

EINVAL ws2 が示すワイド文字列に、照合手順に定義されていないワイド文字コードが含まれている。

ENOSYS 関数はサポートされていない。

使用法 変換処理は、変換後の任意の 2 つのワイド文字列を、wcscmp() または wscmp() がプログラムのロケール ( LC\_COLLATE カテゴリ ) 中の照合手順情報に従って正しく順序付けできるように行われます。

n が 0 であれば ws1 は NULL ポインタでもよい、という規則を利用して、変換前に ws1 の配列の大きさを調べることができます。

wcsxfrm() と wsxfrm() 関数は、setlocale(3C) がロケール変更で呼び出されない限り、マルチスレッドのアプリケーションでは安全に使用できます。

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	例外付き MT-safe
CSI	対応済み

wsxfrm(3C)

関連項目 | setlocale(3C), wscmp(3C), wscoll(3C), wscmp(3C), wscoll(3C),  
attributes(5)

wsxfrm(3C)