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: 819–1213–12
2006 年 11 月

原典: man pages section 3 : Basic Library Functions

Part No: 816-5168-11

Revision A

061213@15821

目次

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。特定のデバイスに関する `ioctl` 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 ioctl は、mtio(7I) のように io 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。

オプション　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。

オペランド　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。

出力　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。

戻り値　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。

エラー　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 errno 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。

使用法　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。

コマンド
修飾子
変数
式
入力文法

使用例　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 example% のプロンプトが出てきます。スーパーユーザーの場合は example# 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。

環境　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。

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す。詳細は `attributes`(5) 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 (bold) で、変数はイタリック体 (Italic) 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。

参照

標準Cライブラリ関数

名前

strftime, cftime, ascftime – 日付と時刻を文字列に変換

形式

```
#include <time.h>

size_t strftime(char *restrict s, size_t maxsize, const char *restrict format,
     const struct tm *restrict timeptr);

int cftime(char *s, char *format, const time_t *clock);

int ascftime(char *s, const char *format, const struct tm *timeptr);
```

機能説明

strftime()、ascftime()、cftime() 関数は、*format* がポイントする文字列の制御に従って、バイトを *s* がポイントする配列に格納します。*format* 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 *s* が示す配列に複写されます。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。strftime() の場合は、*maxsize* を超えないバイト数の文字だけが配列に格納されます。

format 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すが、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。

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 *timeptr* がポイントする構造体に入っている値、cftime() の *clock* で表されている時間で決まります。

%% % 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)

デフォルト

%c 日時の表現 (ロケール固有)

```
%a %b %d %H:%M:%S %Y
```

これは、Solaris 2.4 より以前に最初にサポートされた標準準拠と同様に、デフォルトの表現です。 standards(5) を参照してください。

標準準拠

%c 日時の表現 (ロケール固有)

```
%a %b %e %H:%M:%S %Y
```

これは、Solaris 2.4 で最初にサポートされて以来、Solaris 10 にいたるまでの標準準拠の表現です。

デフォルト　%C　date(1) が出力する形式の日時の表現

これは、Solaris 2.4 より以前に最初にサポートされた標準準拠と同様に、デフォルトの表現です。

標準準拠　%C　世紀を 01 から 99 までの数値で表現 (年を 100 で割って整数に切り捨て)

これは、Solaris 2.4 で最初にサポートされて以来、Solaris 10 にいたるまでの標準準拠の表現です。

%d　日 (01 から 31)。

%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

%F　%Y-%m-%d (ISO 8601:2000 標準の日付形式) と同じ

%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 (00 から 23)。

%I　12 時間表示で表した時 (01 から 12)。

%j　年の通算日 (001 から 366)。

%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 (01 から 12)。

%M　分 (00 から 59)。

%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 60)。値の範囲が (00 から 59) ではなく、(00 から 60) であるのは、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)。月曜日が 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　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 (00 から 99)

%Y　西暦年の完全表示 (たとえば 1993)

%z　ISO 8601:2000 標準形式 (+hhmm または -hhmm) の UTC からのオフセットに置き換わります。タイムゾーンがわからない場合には、文字列には置き換えられません。たとえば、"-0430" は、UTC から 4 時間 30 分おくれ (グリニッジから西に離れていること) を意味します。tm_isdst が 0 の場合には、標準時間のオフセットが使用されます。tm_isdst が正の値の場合には、夏時間のオフセットが使用されます。tm_isdst が負の値の場合には、文字列は返されません。

%Z　タイムゾーン名または省略形 (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
%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、strftime()、cftime()、ascftime() は米国英語で表示されます。setlocale() で LC_TIME カテゴリのロケールを設定すると、strftime()、cftime()、ascftime() の実行結果を特定の言語で出力できるようになります。

時間帯

tzset(3C) 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

`strftime()`、`cftime()`、`ascftime()` 関数は、*s* 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 NULL 文字は含まれません。生成された文字の合計数 (終了 NULL 文字を含む) が *maxsize* を超えていると、`strftime()` は、0 を返し、配列の内容は未確定になります。

使用例

例 1 `strftime()` 関数の例

ここでは、POSIX ロケールの `strftime()` の使用例を示します。*tmptr* 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 `str` の内容がどうなるかを示します。

```
strftime (str, strsize, "%A %b %d %j", tmptr)
```

これにより、`str` の内容は `"Thursday Aug 28 240"` になります。

属性

以下の属性については、`attributes`(5) を参照してください。

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

関連項目

`date`(1), `ctime`(3C), `mktime`(3C), setlocale(3C), `strptime`(3C), `tzset`(3C), `TIMEZONE`(4), `zoneinfo`(4), `attributes`(5), `environ`(5), `standards`(5)

注意事項

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、新しい年の日数が 4 日未満含まれている場合、その週は前年の第 53 週となっていました。しかし、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。

`%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。これらの仕様は変更される可能性があります。ISO C9x 標準がこれらの仕様と異なる決定をした場合、仕様は ISO C9x 標準の決定に準拠して変更されます。

Solaris 8 リリースでは、`%u` の変換仕様が変更されました。この変更は、XPG4 仕様に基づいたものです。

`%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

```
#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);

#include <libintl.h>
#include <locale.h>

char *dcgettext(const char *domainname, const char *msgid, int category);
```

GNU 互換

```
#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。*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。

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 *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)

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);

#include <libintl.h>
#include <locale.h>

char *dcgettext(const char *domainname, const char *msgid, int category);
```

GNU 互換

```
#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。*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。

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 *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)

名前

strftime, cftime, ascftime – 日付と時刻を文字列に変換

形式

```
#include <time.h>

size_t strftime(char *restrict s, size_t maxsize, const char *restrict format,
     const struct tm *restrict timeptr);

int cftime(char *s, char *format, const time_t *clock);

int ascftime(char *s, const char *format, const struct tm *timeptr);
```

機能説明

strftime()、ascftime()、cftime() 関数は、*format* がポイントする文字列の制御に従って、バイトを *s* がポイントする配列に格納します。*format* 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 *s* が示す配列に複写されます。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。strftime() の場合は、*maxsize* を超えないバイト数の文字だけが配列に格納されます。

format 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すが、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。

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 *timeptr* がポイントする構造体に入っている値、cftime() の *clock* で表されている時間で決まります。

%% % 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)

デフォルト

%c 日時の表現 (ロケール固有)

```
%a %b %d %H:%M:%S %Y
```

これは、Solaris 2.4 より以前に最初にサポートされた標準準拠と同様に、デフォルトの表現です。 standards(5) を参照してください。

標準準拠

%c 日時の表現 (ロケール固有)

```
%a %b %e %H:%M:%S %Y
```

		これは、Solaris 2.4 で最初にサポートされて以来、Solaris 10 にいたるまでの標準準拠の表現です。
デフォルト	%C	date(1) が出力する形式の日時の表現
		これは、Solaris 2.4 より以前に最初にサポートされた標準準拠と同様に、デフォルトの表現です。
標準準拠	%C	世紀を 01 から 99 までの数値で表現 (年を 100 で割って整数に切り捨て)
		これは、Solaris 2.4 で最初にサポートされて以来、Solaris 10 にいたるまでの標準準拠の表現です。
	%d	日 (01 から 31)。
	%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
	%F	%Y-%m-%d (ISO 8601:2000 標準の日付形式) と同じ
	%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 (00 から 23)。
	%I	12 時間表示で表した時 (01 から 12)。
	%j	年の通算日 (001 から 366)。
	%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 (01 から 12)。
	%M	分 (00 から 59)。
	%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 60)。値の範囲が (00 から 59) ではなく、(00 から 60) であるのは、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)。月曜日が 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　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 (00 から 99)

%Y　西暦年の完全表示 (たとえば 1993)

%z　ISO 8601:2000 標準形式 (+hhmm または -hhmm) の UTC からのオフセットに置き換わります。タイムゾーンがわからない場合には、文字列には置き換えられません。たとえば、"-0430" は、UTC から 4 時間 30 分おくれ (グリニッジから西に離れていること) を意味します。tm_isdst が 0 の場合には、標準時間のオフセットが使用されます。tm_isdst が正の値の場合には、夏時間のオフセットが使用されます。tm_isdst が負の値の場合には、文字列は返されません。

%Z　タイムゾーン名または省略形 (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
%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、strftime()、cftime()、ascftime() は米国英語で表示されます。setlocale() で LC_TIME カテゴリのロケールを設定すると、strftime()、cftime()、ascftime() の実行結果を特定の言語で出力できるようになります。

時間帯　tzset(3C) 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

`strftime()`、`cftime()`、`ascftime()` 関数は、*s* 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 NULL 文字は含まれません。生成された文字の合計数 (終了 NULL 文字を含む) が *maxsize* を超えていると、`strftime()` は、0 を返し、配列の内容は未確定になります。

使用例

例 1 `strftime()` 関数の例

ここでは、POSIX ロケールの `strftime()` の使用例を示します。*tmptr* 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 `str` の内容がどうなるかを示します。

```
strftime (str, strsize, "%A %b %d %j", tmptr)
```

これにより、`str` の内容は `"Thursday Aug 28 240"` になります。

属性

以下の属性については、`attributes`(5) 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
インタフェースの安定性	`strftime()` が標準

関連項目

`date`(1), `ctime`(3C), `mktime`(3C), `setlocale(3C)`, `strptime`(3C), `tzset`(3C), `TIMEZONE`(4), `zoneinfo`(4), `attributes`(5), `environ`(5), `standards`(5)

注意事項

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、新しい年の日数が 4 日未満含まれている場合、その週は前年の第 53 週となっていました。しかし、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。

`%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。これらの仕様は変更される可能性があります。ISO C9x 標準がこれらの仕様と異なる決定をした場合、仕様は ISO C9x 標準の決定に準拠して変更されます。

Solaris 8 リリースでは、`%u` の変換仕様が変更されました。この変更は、XPG4 仕様に基づいたものです。

`%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

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

これらの関数は、EUC ロケールだけに機能します。

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

これらの関数は、EUC ロケールだけに機能します。

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

これらの関数は、EUC ロケールだけに機能します。

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

これらの関数は、EUC ロケールだけに機能します。

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

ctype, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

`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、`<stdio.h>` 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。

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`) 。

「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 `<ctype.h>` 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。

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、 isupper() または islower() が真である文字に対してのみ、 isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。 standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 islower() 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、 isupper() 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。
	islower()	任意の小文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 isupper() 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、 islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、 isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、 isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト　`isprint()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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　`isprint()`　`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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　`isgraph()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。

標準準拠　`isgraph()`　`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`　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

これらのマクロと関数は、`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
インタフェースの安定性	標準

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

```
#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);

#include <libintl.h>
#include <locale.h>

char *dcgettext(const char *domainname, const char *msgid, int category);
```

GNU 互換

```
#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。*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。

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 *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)

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);

#include <libintl.h>
#include <locale.h>

char *dcgettext(const char *domainname, const char *msgid, int category);
```

GNU 互換

```
#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。*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。

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 *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)

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);

#include <libintl.h>
#include <locale.h>

char *dcgettext(const char *domainname, const char *msgid, int category);
```

GNU 互換

```
#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。*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。

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 *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)

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);

#include <libintl.h>
#include <locale.h>

char *dcgettext(const char *domainname, const char *msgid, int category);
```

GNU 互換

```
#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。*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。

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 *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)

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)`

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

fgetwc – ストリームからワイド文字コードを読み込む

形式

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

wint_t fgetwc(FILE*stream);
```

機能説明

fgetwc() 関数は、*stream* 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。

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。

fgetwc() 関数は、*stream* に対応するファイルの st_atime 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 *stream* を使ったものが最初に正常終了したとき、st_atime フィールドは更新用にマークされます。

戻り値

正常終了時、fgetwc() 関数は *stream* 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 wint_t 型に変換して返します。

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 指示子が設定されて fgetwc() は WEOF を返します。

標準に準拠するアプリケーションの場合 (standards(5) を参照)、ストリーム用の EOF 指示子が設定されると、ストリームが EOF にあるかどうかにかかわらず、fgetwc() は WEOF を返します。

読み込みエラーが発生すると、ストリーム用のエラー指示子が設定され、fgetwc() は WEOF を返して errno をエラーを示す値に設定します。

コード化エラーが発生すると、ストリーム用のエラー指示子が設定され、fgetwc() は WEOF を返して errno をエラーを示す値に設定します。

エラー

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。

EAGAIN
: *stream* 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 fgetwc() 動作が待たされる可能性がある

EBADF
: *stream* 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

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

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

EOVERFLOW　ファイルは通常ファイルで、対応する *stream* 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

fgetwc() 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　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
インタフェースの安定性	標準

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), standards(5)

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 <widec.h>

wchar_t *getws(wchar_t *ws);

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

wchar_t *fgetws(wchar_t *restrict ws, int n, FILE *restrict 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 ungetc(3C) または ungetwc(3C) への呼び出しによって与えられたものではないデータを返す *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。標準に準拠するアプリケーションの場合 (standards(5) を参照)、ストリーム用の EOF 指示子が設定されると、ストリームが EOF にあるかどうかにかかわらず、fgetws() はヌル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
インタフェースの安定性	`fgetws()` が標準

関連項目

`ferror`(3C), `fgetwc`(3C),`fread`(3C), `getwc`(3C), `putws`(3C), `scanf`(3C), `ungetc`(3C), `ungetwc`(3C), `attributes`(5), `standards`(5)

名前

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

形式

```
#include <stdio.h>

int printf(const char *restrict format, /* args*/ ...);

int fprintf(FILE *restrict stream, const char *restrict format, /* args*/ ...);

int sprintf(char *restrict s, const char *restrict format, /* args*/ ...);

int snprintf(char *restrict s, size_t n, const char *restrict 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* を追加したのと同等です。*n* には *s* によって参照されるバッファーのサイズが指定されます。*n* が 0 の場合、配列には何も書き込まれず、*s* はヌルポインタになる可能性もあります。そうでない場合、*n*-1 番目までの出力バイトが配列に書き込まれ、*n* 番目以降の出力バイトが破棄されます。そして、実際に配列に書き込まれたバイトの終わりにヌルバイトが書き込まれます。

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、使われません。

変換指定の書式 % の代わりに書式 %*n*$ を使用すると、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。

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。

  変換指定文字が 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。a、A、e、E、f 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指定文字で表される場合、動作は未定義です。

  変換指定文字が 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す。

  %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、その後に 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。

- 長さ修飾子 (任意)。引数のサイズを指定します。
- 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\
", 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、%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。 a、A、e、E、f、F、g、および G 変換の場合、必ず、結果には小数点が付きます。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, a, A, e, E, f, 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。

長さ修飾子

次に、長さ修飾子とその意味を示します。

`hh` 後続の変換文字 d、i、o、u、x、および X が signed char 型または unsigned char 型の引数に適用されることを指定します (引数は整数昇格の規則に従って昇格されますが、値は signed char または unsigned char の形式に変換されてから表示されます)。あるいは、後続の変換文字 n が signed char 型の引数へのポインタに適用されることを指定します。

`h` 後続の変換文字 d、i、o、u、x、および X が short 型または unsigned short 型の引数に適用されることを指定します (引数は整数昇格の規則に従って昇格されますが、値は short または unsigned short の形式に変換されてから表示されます)。あるいは、後続の変換文字 n が short 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) 後続の変換文字 d、i、o、u、x、または X が long 型あるいは unsigned long 型の引数に適用されることを指定します。あるいは、後続の変換文字 n が 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。あるいは、後続の変換文字 c が wint_t 型の引数に適用されることを指定します。あるいは、後続の変換文字 s が wchar_t 型の引数へのポインタに適用されることを指定します。あるいは、後続の変換文字 a、A、e、E、f、F、g、または G には効果がないことを指定します。

`ll` (小文字のエルと小文字のエル) 後続の変換文字 d、i、o、u、x、または X が long long 型あるいは unsigned long long 型の引数に適用されることを指定します。あるいは、後続の変換文字 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。

`j` 後続の変換文字 d、i、o、u、x、または X が intmax_t 型あるいは uintmax_t 型の引数に適用されることを指定します。あるいは、後続の変換文字 n が intmax_t 型の引数へのポインタに適用されることを指定します。

z　後続の変換文字 d、i、o、u、x または X が size_t 型あるいは対応する符号付き整数型の引数に適用されることを指定します。あるいは、後続の変換文字 n が size_t 型の引数に対応する符号付き整数型の引数へのポインタに適用されることを指定します。

t　後続の変換文字 d、i、o、u、x または X が ptrdiff_t 型あるいは対応する符号なし整数型の引数に適用されることを指定します。あるいは、後続の変換文字 n が ptrdiff_t 型の引数へのポインタに適用されることを指定します。

L　後続の変換文字 a、A、e、E、f、F、g、または G が long double 型の引数に適用されることを指定します。

長さ修飾子を上記以外の変換文字と一緒に指定した場合、その動作は未定義です。

変換指定文字

各変換指定文字を指定すると、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。

変換指定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 変換指定文字と同じ動作です。

f, F　　double の引数は、[–]*ddd*.*ddd* の形式で 10 進数に変換されます。小数点文字 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 に指定し、# フラグを指定しない場合、小数点文字は出力されません。小数点文字が出力されると、1 個以上の数字がその前に付きます。変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。

変換文字 f の場合、無限または非数を表す double 型の引数は、変換文字 e の形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “infinity” または “Infinity” と表示され、そうでない場合は “inf” または “Inf” と表示されます。

変換文字 F の場合、無限または非数を表す double 型の引数は、変換文字 E の SUSv3 形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “INFINITY” と表示され、そうでない場合は “INF” と表示されます。

e,E　　double の引数は、[–]*d*.*ddd*e±*dd* の形式に変換されます。小数点文字の前には 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)。小数点文字の後には、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、小数点文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。変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。値は適切な桁数に丸められます。

無限または非数の値は、次のように処理されます。

SUSv3　　変換文字 e の場合、無限を表す double 型の引数は、変換の精度が 7 以上の場合は “]&–]infinity” と表示され、そうでない場合は「[–]inf」と表示されます。非数を表す double 型の引数は、「[–]nan」と表示されます。変換文字 E の場合、「infinity」、「inf」、および「nan」の代わりに、それぞれ、「INFINITY」、「INF」、および「NAN」と表示されます。符号の印刷は、上記の規則に従います。

デフォルト　　無限を表す double 型の引数は、変換の精度が 7 以上の場合は “]&–]Infinity” と表示され、そうでない場合は “]&–]Inf” と表示されます。非数を表す double 型の引数は、“]&–]NaN” と表示されます。符号の印刷は、上記の規則に従います。

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。小数点文字は、その後に数字が続く場合だけ出力されます。

無限または非数を表す double 型の引数は、変換文字 e または E の形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “infinity”、“INFINITY”、または “Infinity” と表示され、そうでない場合は “inf”、“INF”、または “Inf” と表示されます。

a, A　　浮動小数点数を表す double 型の引数は、[-]0*xh.hhhhp±d* の形式に変換されます。基数点の前にある 1 桁の 16 進数 (h) は、変換された値が 0 の場合は 0 になり、そうでない場合は 1 になります。基数点の後にある 16 進数 (hhhh) の桁数は精度と同じになります。精度を指定しなかった場合、基数点の後にある 16 進数の桁数は、double 型の値を変換するときは 13 になり、long double 型の値を変換するときは 16 (x86 の場合) または 28 (SPARC の場合) になります。精度に 0 を指定し、# フラグを指定しなかった場合、10 進小数点文字は表示されません。変換文字 a の場合は文字「abcdef」が使用され、変換文字 A の場合は文字「ABCDEF」が使用されます。変換文字 A を指定した場合、「x」と「p」の代わりに、それぞれ、「X」と「P」が表示されます。指数の桁数は必ず 1 桁が表示され、2 の 10 進指数を表すのに必要な桁まで表示されます。値が 0 の場合、指数は 0 になります。

変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。

無限または非数を表す double 型の引数は、変換文字 e または E の SUSv3 形式に変換されます。

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、精度は付けないで、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、wcrtomb(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、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。

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() 関数は *n* が十分な大きさであったと仮定した場合に *s* に書き込まれる (はずの) バイト数を、末尾のヌルバイト (の分) を含まない値で返します。 snprintf() への呼び出しで *n* の値が 0 の場合、書き込まれるバイト数を返し、このとき *s* はヌルポインタとすることができます。

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。

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。

\　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

\\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\
"
```

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\
"
```

生成されるメッセージは次のとおりです。

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

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。 (続き)

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	下記参照
CSI	対応済み
インタフェースの安定性	標準

sprintf() 関数と snprintf() 関数は、「非同期シグナル安全」です。printf() 関数と fprintf() 関数は、setlocale(3C) を呼び出してロケールを変更していない限り、マルチスレッドアプリケーションで安全に使用できます。

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), vprintf(3C), wcstombs(3C), wctomb(3C), attributes(5), environ(5), standards(5)

注意事項

Solaris 10 より前のリリースで c89 を使用してコンパイルした 32 ビットアプリケーションでは、長さ修飾子 j を使用した場合の動作は未定義です。

n = 0 のときに snprintf() が返す値は、Solaris 10 リリースで変更されました。この変更は、SUSv3 仕様に基づいています。以前の動作は初期の SUSv2 仕様に基づいており、*n* = 0 のときに snprintf() が返す値は 1 よりも小さな未指定の値でした。

名前

fputwc, putwc, putwchar – ワイド文字コードをストリームへ書き出す

形式

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

wint_t fputwc(wchar_t wc, FILE*stream);

wint_t putwc(wchar_t wc, FILE*stream);

#include <wchar.h>

wint_t putwchar(wchar_t wc);
```

機能説明

fputwc( )

fputwc() 関数は、ワイド文字コード *wc* に対応する文字を、*stream* 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。

ファイル中の st_ctime と st_mtime の両フィールドは、fputwc() 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。

putwc( )

putwc() 関数は、マクロとして実装される点を除いては、fputwc() 関数と同じです。

putwchar( )

putwchar(*wc*) の呼び出しは、putwc(*wc*, stdout) と同じです。putwchar() ルーチンはマクロとして実装されます。

戻り値

正常終了時、fputwc()、putwc()、および putwchar() は *wc* 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ー中のデータを書き出す必要があるとき、fputwc()、putwc()、および putwchar() 関数は以下の条件で異常終了します。

EAGAIN — *stream* 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 — *stream* 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 *stream* 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
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 シグナルも呼び出し元スレッドへ送られる

fputwc()、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 fputwc() 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
インタフェースの安定性	標準

関連項目

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

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 *restrict s, FILE *restrict 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
インタフェースの安定性	標準

関連項目

exit(2), abort(3C), fclose(3C), fflush(3C), fopen(3C), fputwc(3C), attributes(5), standards(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 *restrict format, ...);

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

int sscanf(const char *restrict s, const char *restrict 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、初期シフトの状態で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)
- 長さ修飾子 (任意)。受け取るオブジェクトのサイズを指定します。
- 適用される変換形式を指定する変換指定文字 (有効な変換指定文字については、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 (入力が不適当) 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、有効な入力が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。入力項目の長さは、指定した最大フィールド幅によって制限されます。同時に、入力項目の長さの単位は、指定した変換文字によって文字またはバイト数のどちらかに解釈されます。scanf() は、入力項目の終わりを判断するために、余分な文字を読み取る必要がある場合があります。デフォルトの Solaris モードでは、入力項目

は「マッチングシーケンスの一部である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、複数の文字を読み取る必要があります。C99/SUSv3 モードでは、入力項目は「マッチングシーケンスと同じ (あるいは、その前半部分) である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、多くても 1 文字を読み取るだけでかまいません。したがって、C99/SUSv3 モードでは、strtod(3C) や strtol(3C) などの関数には受け入れられるシーケンスが scanf() では受け入れられない場合があります。どちらのモードでも、scanf() は、ungetc(3C) を使用して、超過バイトの読み取りをプッシュバックしようとします。このような試みがすべて成功したと想定した場合、入力項目の後に続く最初のバイト (もしあれば) は、読み取られずに残ります。入力項目の長さが 0 の場合、変換は失敗します。この状態はマッチングエラーです。ただし、ファイルの終わり、コード化エラー、または、読み取りエラーによって、ストリームからの入力が妨害された場合は入力エラーです。

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、変換文字に適当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当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。

長さ修飾子

次に、長さ修飾子とその意味を示します。

hh	後続の変換文字 d、i、o、u、x、X または n が signed char あるいは unsigned char へのポインタ型を持つ引数に適用されることを指定します。
h	後続の変換文字 d、i、o、u、x、X、または n が short あるいは unsigned short へのポインタ型を持つ引数に適用されることを指定します。
l (小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long あるいは unsigned long へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 a、A、e、E、f、F、g、または G が double へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 c、s、または [が wchar_t へのポインタ型を持つ引数に適用されることを指定します。
ll (小文字のエルと小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long long あるいは unsigned long long へのポインタ型を持つ引数に適用されることを指定します。

j　後続の変換文字 d、i、o、u、x、X、または n が intmax_t あるいは uintmax_t へのポインタ型を持つ引数に適用されることを指定します。

z　後続の変換文字 d、i、o、u、x、X、または n が size_t あるいは対応する符号付き整数型へのポインタ型を持つ引数に適用されることを指定します。

t　後続の変換文字 d、i、o、u、x、X、または n が ptrdiff_t あるいは対応する unsigned 型へのポインタ型を持つ引数に適用されることを指定します。

L　後続の変換文字 a、A、e、E、f、F、g、または G が long double へのポインタ型を持つ引数に適用されることを指定します。

長さ修飾子を上記以外の変換文字と一緒に指定した場合、その動作は未定義です。

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、strtol(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、strtol() 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。

a,e,f,g　符号付き浮動小数点数、無限、または NaN 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。変換文字 e、f、および g は、C99/SUSv3 モード (standards(5)

を参照) の場合だけ、16 進数の浮動小数点値に一致します。しかし、変換文字 a は常に、16 進数の浮動小数点値に一致します。

これらの変換文字は、strtod(3C) が受け入れる変換対象シーケンス (INF、INFINITY、NAN、および NAN (*n-char-sequence*) の形式を含む) にマッチングします。変換の結果は、マッチングシーケンスとともに strtod() (あるいは、strtof() または strtold()) を呼び出したときと同じで、浮動小数点の例外が発生して、(可能であれば)、errno に ERANGE が設定されます。

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。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 A、E、F、G、および X はそれぞれ a、e、f、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等です。

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 呼び出し

```
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 呼び出し

```
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
```

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
インタフェースの安定性	標準

関連項目

`fgetc`(3C), `fgets`(3C), `fgetwc`(3C), `fread`(3C), `isspace`(3C), `printf`(3C), `setlocale`(3C), `strtod`(3C), `strtol`(3C), `strtoul`(3C), `wcrtomb`(3C), `ungetc`(3C), `attributes`(5), `standards`(5)

名前

getdate – ユーザー形式の日付および時刻の変換

形式

```
#include <time.h>

struct tm *getdate(const char *string);

extern int getdate_err;
```

機能説明

getdate() 関数は、*string* 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 <time.h> ヘッダーにあります。

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。

変換仕様

以下の変換仕様がサポートされています。

%%	% 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	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
%d	日 (1 から 31)。10 の位を示す 0 の指定は任意
%D	%m/%d/%y 形式で表した日付
%e	%d と同じ
%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)。10 の位を示す 0 の指定は任意
%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
%j	年の通算日 (1 から 366)。10 の位を示す 0 の指定は任意
%m	月を示す数値 (1から 12)。10 の位を示す 0 の指定は任意

%M 分 (0 から 59)。10 の位を示す 0 の指定は任意

%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

SUSv3

%S 秒 (0 から 60)。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 60) であるのは、うるう秒に対処するため

デフォルトとその他の標準

%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)。週は日曜日から始まるとする。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)。週は月曜日から始まるとする。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
%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ければ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。

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。

例 1 getdate() 関数の例　　(続き)

```
%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")
```

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。

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

関連項目

ctype(3C), mktime(3C), setlocale(3C), strftime(3C), strptime(3C), attributes(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

```
#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);

#include <libintl.h>
#include <locale.h>

char *dcgettext(const char *domainname, const char *msgid, int category);
```

GNU 互換

```
#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。*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。

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 *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)

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
インタフェースの安定性	標準

関連項目

fgetwc(3C), attributes(5), standards(5)

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
インタフェースの安定性	標準

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

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 <widec.h>

wchar_t *getws(wchar_t *ws);

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

wchar_t *fgetws(wchar_t *restrict ws, int n, FILE *restrict 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 ungetc(3C) または ungetwc(3C) への呼び出しによって与えられたものではないデータを返す *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。標準に準拠するアプリケーションの場合 (standards(5) を参照)、ストリーム用の EOF 指示子が設定されると、ストリームが EOF にあるかどうかにかかわらず、fgetws() はヌル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
インタフェースの安定性	`fgetws()` が標準

関連項目

`ferror`(3C), `fgetwc`(3C),`fread`(3C), `getwc`(3C), `putws`(3C), `scanf`(3C), `ungetc`(3C), `ungetwc`(3C), `attributes`(5), `standards`(5)

名前

iconv – コード変換関数

形式

Default

```
#include <iconv.h>

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

SUSv3

```
#include <iconv.h>

size_t iconv(iconv_t cd, char **restrict inbuf, size_t *restrict inbytesleft, char
     **restrict outbuf, size_t *restrict 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 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	cd 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; \
```

例 1 iconv() 関数の使用　　*(続き)*

```
        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;


    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.
             */

            /*
```

例 1 iconv() 関数の使用　(続き)

```
             * 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
                 */
                (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);
```

例 1 iconv() 関数の使用　　*(続き)*

```
            /*
             * 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.
             */
            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);
```

例 1 iconv() 関数の使用　(続き)

```
            } 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`
　32 ビット用の変換モジュール

`/usr/lib/iconv/sparcv9/*.so`
　64 ビット SPARC 用の変換モジュール

`/usr/lib/iconv/amd64/*.so`
　64 ビット amd64 用の変換モジュール

`/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
インタフェースの安定性	標準

関連項目

`geniconvtbl`(1), `iconv`(1), `iconv_close`(3C), `iconv_open`(3C), `geniconvtbl`(4), `attributes`(5), `iconv`(5), `iconv_unicode`(5), `standards`(5)

名前

ctype, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

`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、`<stdio.h>` 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。

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`) 。

「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 `<ctype.h>` 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。

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、 isupper() または islower() が真である文字に対してのみ、 isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、 SUSv3 より前の標準に準拠するアプリケーションでは利用できません。 standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 islower() 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、 isupper() 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。
	islower()	任意の小文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 isupper() 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、 islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、 isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、 isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト	`isprint()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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	`isprint()`	`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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	`isgraph()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。
標準準拠	`isgraph()`	`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`	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

これらのマクロと関数は、`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
インタフェースの安定性	標準

関連項目

`setlocale(3C)`, `stdio`(3C), `ascii`(5), `environ`(5), `standards`(5)

名前

ctype, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

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、<stdio.h> 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。

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) 。

「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 <ctype.h> 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。

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、isupper() または islower() が真である文字に対してのみ、isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl()、isdigit()、ispunct()、isspace()、islower() 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、isupper() 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。
	islower()	任意の小文字、または iscntrl()、isdigit()、ispunct()、isspace()、isupper() 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、islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト　`isprint()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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　`isprint()`　`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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　`isgraph()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。

標準準拠　`isgraph()`　`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`　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

これらのマクロと関数は、`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
インタフェースの安定性	標準

関連項目

`setlocale(3C)`, `stdio`(3C), `ascii`(5), `environ`(5), `standards`(5)

名前

ctype, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

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、<stdio.h> 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。

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) 。

「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 <ctype.h> 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。

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、isupper() または islower() が真である文字に対してのみ、isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl() 、isdigit() 、ispunct() 、isspace() 、islower() 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、isupper() 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。
	islower()	任意の小文字、または iscntrl() 、isdigit() 、ispunct() 、isspace() 、isupper() 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、islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト	`isprint()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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	`isprint()`	`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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	`isgraph()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。
標準準拠	`isgraph()`	`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`	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

これらのマクロと関数は、`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
インタフェースの安定性	標準

関連項目

`setlocale(3C)`, `stdio`(3C), `ascii`(5), `environ`(5), `standards`(5)

名前

ctype, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

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、<stdio.h> 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。

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) 。

「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 <ctype.h> 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。

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、isupper() または islower() が真である文字に対してのみ、isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl()、isdigit()、ispunct()、isspace()、islower() 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、isupper() 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。
	islower()	任意の小文字、または iscntrl()、isdigit()、ispunct()、isspace()、isupper() 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、islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト　`isprint()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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　`isprint()`　`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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　`isgraph()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。

標準準拠　`isgraph()`　`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`　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

これらのマクロと関数は、`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
インタフェースの安定性	標準

関連項目

`setlocale(3C)`, `stdio`(3C), `ascii`(5), `environ`(5), `standards`(5)

名前

ctype, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

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、<stdio.h> 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。

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) 。

「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 <ctype.h> 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。

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、 isupper() または islower() が真である文字に対してのみ、 isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。 standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 islower() 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、 isupper() 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。
	islower()	任意の小文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 isupper() 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、 islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、 isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、 isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト	`isprint()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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	`isprint()`	`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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	`isgraph()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。
標準準拠	`isgraph()`	`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`	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

これらのマクロと関数は、`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
インタフェースの安定性	標準

関連項目

`setlocale(3C)`, `stdio`(3C), `ascii`(5), `environ`(5), `standards`(5)

名前

ctype, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

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、<stdio.h> 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。

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) 。

「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 <ctype.h> 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。

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、 isupper() または islower() が真である文字に対してのみ、 isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。 standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 islower() 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、 isupper() 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。
	islower()	任意の小文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 isupper() 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、 islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、 isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、 isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト　`isprint()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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　`isprint()`　`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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　`isgraph()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。

標準準拠　`isgraph()`　`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`　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

これらのマクロと関数は、`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
インタフェースの安定性	標準

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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc*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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

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、<stdio.h> 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。

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) 。

「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 <ctype.h> 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。

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、isupper() または islower() が真である文字に対してのみ、isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl() 、isdigit() 、ispunct() 、isspace() 、islower() 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、isupper() 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。
	islower()	任意の小文字、または iscntrl() 、isdigit() 、ispunct() 、isspace() 、isupper() 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、islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト	`isprint()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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	`isprint()`	`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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	`isgraph()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。
標準準拠	`isgraph()`	`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`	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

これらのマクロと関数は、`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
インタフェースの安定性	標準

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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc*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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

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、<stdio.h> 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。

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) 。

「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 <ctype.h> 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。

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、isupper() または islower() が真である文字に対してのみ、isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl()、isdigit()、ispunct()、isspace()、islower() 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、isupper() 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。
	islower()	任意の小文字、または iscntrl()、isdigit()、ispunct()、isspace()、isupper() 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、islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト　`isprint()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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　`isprint()`　`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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　`isgraph()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。

標準準拠　`isgraph()`　`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`　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

これらのマクロと関数は、`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
インタフェースの安定性	標準

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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc*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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc*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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

`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、`<stdio.h>` 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。

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`) 。

「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 `<ctype.h>` 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。

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、 isupper() または islower() が真である文字に対してのみ、 isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。 standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 islower() 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、 isupper() 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。
	islower()	任意の小文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 isupper() 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、 islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、 isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、 isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト	`isprint()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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	`isprint()`	`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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	`isgraph()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。
標準準拠	`isgraph()`	`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`	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

これらのマクロと関数は、`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
インタフェースの安定性	標準

関連項目

`setlocale(3C)`, `stdio`(3C), `ascii`(5), `environ`(5), `standards`(5)

名前

ctype, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

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、<stdio.h> 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。

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) 。

「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 <ctype.h> 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。

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、isupper() または islower() が真である文字に対してのみ、isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl()、isdigit()、ispunct()、isspace()、islower() 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、isupper() 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。
	islower()	任意の小文字、または iscntrl()、isdigit()、ispunct()、isspace()、isupper() 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、islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト　`isprint()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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　`isprint()`　`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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　`isgraph()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。

標準準拠　`isgraph()`　`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`　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

これらのマクロと関数は、`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
インタフェースの安定性	標準

関連項目

`setlocale(3C)`, `stdio`(3C), `ascii`(5), `environ`(5), `standards`(5)

名前

ctype, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

`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、`<stdio.h>` 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。

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`) 。

「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 `<ctype.h>` 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。

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、 isupper() または islower() が真である文字に対してのみ、 isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。 standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 islower() 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、 isupper() 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。
	islower()	任意の小文字、または iscntrl() 、 isdigit() 、 ispunct() 、 isspace() 、 isupper() 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、 islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、 isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、 isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト	`isprint()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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	`isprint()`	`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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	`isgraph()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。
標準準拠	`isgraph()`	`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`	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

これらのマクロと関数は、`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
インタフェースの安定性	標準

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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc*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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

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、<stdio.h> 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。

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) 。

「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 <ctype.h> 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。

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、isupper() または islower() が真である文字に対してのみ、isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl()、isdigit()、ispunct()、isspace()、islower() 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、isupper() 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。
	islower()	任意の小文字、または iscntrl()、isdigit()、ispunct()、isspace()、isupper() 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、islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト　`isprint()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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　`isprint()`　`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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　`isgraph()`　`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。

標準準拠　`isgraph()`　`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`　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

これらのマクロと関数は、`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
インタフェースの安定性	標準

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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc*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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc*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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc*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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc*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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	iswctype(*wc*, `wctype("alnum")`)
iswalpha(*wc*)	iswctype(*wc*, `wctype("alpha")`)
iswcntrl(*wc*)	iswctype(*wc*, `wctype("cntrl")`)
iswdigit(*wc*)	iswctype(*wc*, `wctype("digit")`)
iswgraph(*wc*)	iswctype(*wc*, `wctype("graph")`)
iswlower(*wc*)	iswctype(*wc*, `wctype("lower")`)
iswprint(*wc*)	iswctype(*wc*, `wctype("print")`)
iswpunct(*wc*)	iswctype(*wc*, `wctype("punct")`)
iswspace(*wc*)	iswctype(*wc*, `wctype("space")`)
iswupper(*wc*)	iswctype(*wc*, `wctype("upper")`)

iswxdigit(*wc*)	iswctype(*wc*, `wctype("xdigit")`)

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
インタフェースの安定性	標準

関連項目

`iswalpha`(3C), `setlocale`(3C), `wctype`(3C), `attributes`(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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc*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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc*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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc*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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc*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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc*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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc*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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc*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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)` プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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 iswalpha(wint_t wc);
```

機能説明

これらの関数は、*wc* 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、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc*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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(*wc*)	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	プログラムの現在のロケールにおいて、*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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(`*wc*`)`	*wc* 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, isalpha, isalnum, isascii, isblank, iscntrl, isdigit, islower, isprint, isspace, isupper, ispunct, isgraph, isxdigit – 文字処理

形式

```
#include <ctype.h>

int isalpha(int c);

int isalnum(int c);

int isascii(int c);

int isblank(int c);

int iscntrl(int c);

int isdigit(int c);

int isgraph(int c);

int islower(int c);

int isprint(int c);

int ispunct(int c);

int isspace(int c);

int isupper(int c);

int isxdigit(int c);
```

機能説明

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。

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、<stdio.h> 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。

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) 。

「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 <ctype.h> 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。

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、isupper() または islower() が真である文字に対してのみ、isalpha() は真を返します。
	isblank()	*c* が現在のロケールにおける空白 (blank) クラスの文字であるかどうかを検査します。このマクロまたは関数は、SUSv3 より前の標準に準拠するアプリケーションでは利用できません。standards(5) を参照してください。
	isupper()	任意の大文字、または iscntrl() 、isdigit() 、ispunct() 、isspace() 、islower() 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、isupper() 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。
	islower()	任意の小文字、または iscntrl() 、isdigit() 、ispunct() 、isspace() 、isupper() 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、islower() 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。
	isdigit()	任意の 10 進数の文字について検査します。
デフォルト	isxdigit()	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	isxdigit()	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
	isalnum()	isalpha() または isdigit() 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。
	isspace()	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 isalnum() 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、isspace() は標準空白文字についてだけ真を返します。
	ispunct()	空白文字でなく、isalnum() または iscntrl() が真でもない印刷文字について検査します。

デフォルト	`isprint()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` 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	`isprint()`	`iscntrl()` が偽であり、`isalnum()`、`isgraph()`、`ispunct()`、空白文字、現在のロケールで定義済みの print 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	`isgraph()`	`ispunct()`、`isupper()`、`islower()`、`isdigit()` が真である任意の文字について検査します。
標準準拠	`isgraph()`	`isalnum()`、`ispunct()` が真である任意の文字、または、空白文字でなく、`iscntrl()` が真でもない現在のロケールで定義済みの graph 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()`	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

これらのマクロと関数は、`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
インタフェースの安定性	標準

関連項目

`setlocale(3C)`, `stdio`(3C), `ascii`(5), `environ`(5), `standards`(5)

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
インタフェースの安定性	標準

関連項目

mbstowcs(3C), mbtowc(3C), setlocale(3C), wcstombs(3C), wctomb(3C), attributes(5), standards(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 *restrict pwcs, const char *restrict 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)

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 *s* 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
インタフェースの安定性	標準

関連項目

mblen(3C), mbstowcs(3C), setlocale(3C), wcstombs(3C), wctomb(3C), attributes(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

```
#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);

#include <libintl.h>
#include <locale.h>

char *dcgettext(const char *domainname, const char *msgid, int category);
```

GNU 互換

```
#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。*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。

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 *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)

名前

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

形式

```
#include <stdio.h>

int printf(const char *restrict format, /* args*/ ...);

int fprintf(FILE *restrict stream, const char *restrict format, /* args*/ ...);

int sprintf(char *restrict s, const char *restrict format, /* args*/ ...);

int snprintf(char *restrict s, size_t n, const char *restrict 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* を追加したのと同等です。*n* には *s* によって参照されるバッファーのサイズが指定されます。*n* が 0 の場合、配列には何も書き込まれず、*s* はヌルポインタになる可能性もあります。そうでない場合、*n*-1 番目までの出力バイトが配列に書き込まれ、*n* 番目以降の出力バイトが破棄されます。そして、実際に配列に書き込まれたバイトの終わりにヌルバイトが書き込まれます。

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、使われません。

変換指定の書式 % の代わりに書式 %*n*$ を使用すると、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。

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。

  変換指定文字が 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。a、A、e、E、f 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指定文字で表される場合、動作は未定義です。

  変換指定文字が 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す。

  %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、その後に 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。

- 長さ修飾子 (任意)。引数のサイズを指定します。
- 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\
", 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、%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。 a、A、e、E、f、F、g、および G 変換の場合、必ず、結果には小数点が付きます。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, a, A, e, E, f, 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。

長さ修飾子

次に、長さ修飾子とその意味を示します。

hh 後続の変換文字 d、i、o、u、x、および X が signed char 型または unsigned char 型の引数に適用されることを指定します (引数は整数昇格の規則に従って昇格されますが、値は signed char または unsigned char の形式に変換されてから表示されます)。あるいは、後続の変換文字 n が signed char 型の引数へのポインタに適用されることを指定します。

h 後続の変換文字 d、i、o、u、x、および X が short 型または unsigned short 型の引数に適用されることを指定します (引数は整数昇格の規則に従って昇格されますが、値は short または unsigned short の形式に変換されてから表示されます)。あるいは、後続の変換文字 n が short 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) 後続の変換文字 d、i、o、u、x、または X が long 型あるいは unsigned long 型の引数に適用されることを指定します。あるいは、後続の変換文字 n が 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。あるいは、後続の変換文字 c が wint_t 型の引数に適用されることを指定します。あるいは、後続の変換文字 s が wchar_t 型の引数へのポインタに適用されることを指定します。あるいは、後続の変換文字 a、A、e、E、f、F、g、または G には効果がないことを指定します。

ll (小文字のエルと小文字のエル) 後続の変換文字 d、i、o、u、x、または X が long long 型あるいは unsigned long long 型の引数に適用されることを指定します。あるいは、後続の変換文字 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。

j 後続の変換文字 d、i、o、u、x、または X が intmax_t 型あるいは uintmax_t 型の引数に適用されることを指定します。あるいは、後続の変換文字 n が intmax_t 型の引数へのポインタに適用されることを指定します。

z　後続の変換文字 d、i、o、u、x または X が size_t 型あるいは対応する符号付き整数型の引数に適用されることを指定します。あるいは、後続の変換文字 n が size_t 型の引数に対応する符号付き整数型の引数へのポインタに適用されることを指定します。

t　後続の変換文字 d、i、o、u、x または X が ptrdiff_t 型あるいは対応する符号なし整数型の引数に適用されることを指定します。あるいは、後続の変換文字 n が ptrdiff_t 型の引数へのポインタに適用されることを指定します。

L　後続の変換文字 a、A、e、E、f、F、g、または G が long double 型の引数に適用されることを指定します。

長さ修飾子を上記以外の変換文字と一緒に指定した場合、その動作は未定義です。

変換指定文字

各変換指定文字を指定すると、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。

変換指定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 変換指定文字と同じ動作です。

f, F　　double の引数は、[–]*ddd*.*ddd* の形式で 10 進数に変換されます。小数点文字 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 に指定し、# フラグを指定しない場合、小数点文字は出力されません。小数点文字が出力されると、1 個以上の数字がその前に付きます。変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。

変換文字 f の場合、無限または非数を表す double 型の引数は、変換文字 e の形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “infinity” または “Infinity” と表示され、そうでない場合は “inf” または “Inf” と表示されます。

変換文字 F の場合、無限または非数を表す double 型の引数は、変換文字 E の SUSv3 形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “INFINITY” と表示され、そうでない場合は “INF” と表示されます。

e,E　　double の引数は、[–]*d*.*ddd*e±*dd* の形式に変換されます。小数点文字の前には 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)。小数点文字の後には、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、小数点文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。変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。値は適切な桁数に丸められます。

無限または非数の値は、次のように処理されます。

SUSv3　　変換文字 e の場合、無限を表す double 型の引数は、変換の精度が 7 以上の場合は “]&–]infinity” と表示され、そうでない場合は「[–]inf」と表示されます。非数を表す double 型の引数は、「[–]nan」と表示されます。変換文字 E の場合、「infinity」、「inf」、および「nan」の代わりに、それぞれ、「INFINITY」、「INF」、および「NAN」と表示されます。符号の印刷は、上記の規則に従います。

デフォルト　　無限を表す double 型の引数は、変換の精度が 7 以上の場合は “]&–]Infinity” と表示され、そうでない場合は “]&–]Inf” と表示されます。非数を表す double 型の引数は、“]&–]NaN” と表示されます。符号の印刷は、上記の規則に従います。

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。小数点文字は、その後に数字が続く場合だけ出力されます。

無限または非数を表す double 型の引数は、変換文字 e または E の形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “infinity”、“INFINITY”、または “Infinity” と表示され、そうでない場合は “inf”、“INF”、または “Inf” と表示されます。

a, A　　浮動小数点数を表す double 型の引数は、[-]0*xh.hhhhp±d* の形式に変換されます。基数点の前にある 1 桁の 16 進数 (h) は、変換された値が 0 の場合は 0 になり、そうでない場合は 1 になります。基数点の後にある 16 進数 (hhhh) の桁数は精度と同じになります。精度を指定しなかった場合、基数点の後にある 16 進数の桁数は、double 型の値を変換するときは 13 になり、long double 型の値を変換するときは 16 (x86 の場合) または 28 (SPARC の場合) になります。精度に 0 を指定し、# フラグを指定しなかった場合、10 進小数点文字は表示されません。変換文字 a の場合は文字「abcdef」が使用され、変換文字 A の場合は文字「ABCDEF」が使用されます。変換文字 A を指定した場合、「x」と「p」の代わりに、それぞれ、「X」と「P」が表示されます。指数の桁数は必ず 1 桁が表示され、2 の 10 進指数を表すのに必要な桁まで表示されます。値が 0 の場合、指数は 0 になります。

変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。

無限または非数を表す double 型の引数は、変換文字 e または E の SUSv3 形式に変換されます。

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、精度は付けないで、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、wcrtomb(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、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。

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() 関数は *n* が十分な大きさであったと仮定した場合に *s* に書き込まれる (はずの) バイト数を、末尾のヌルバイト (の分) を含まない値で返します。 snprintf() への呼び出しで *n* の値が 0 の場合、書き込まれるバイト数を返し、このとき *s* はヌルポインタとすることができます。

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。

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。

\　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

\\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\
"
```

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\
"
```

生成されるメッセージは次のとおりです。

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

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。 *(続き)*

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	下記参照
CSI	対応済み
インタフェースの安定性	標準

sprintf() 関数と snprintf() 関数は、「非同期シグナル安全」です。printf() 関数と fprintf() 関数は、setlocale(3C) を呼び出してロケールを変更していない限り、マルチスレッドアプリケーションで安全に使用できます。

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), vprintf(3C), wcstombs(3C), wctomb(3C), attributes(5), environ(5), standards(5)

注意事項

Solaris 10 より前のリリースで c89 を使用してコンパイルした 32 ビットアプリケーションでは、長さ修飾子 j を使用した場合の動作は未定義です。

$n = 0$ のときに snprintf() が返す値は、Solaris 10 リリースで変更されました。この変更は、SUSv3 仕様に基づいています。以前の動作は初期の SUSv2 仕様に基づいており、$n = 0$ のときに snprintf() が返す値は 1 よりも小さな未指定の値でした。

名前　fputwc, putwc, putwchar – ワイド文字コードをストリームへ書き出す

形式

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

wint_t fputwc(wchar_t wc, FILE*stream);

wint_t putwc(wchar_t wc, FILE*stream);

#include <wchar.h>

wint_t putwchar(wchar_t wc);
```

機能説明

fputwc( )　fputwc( ) 関数は、ワイド文字コード *wc* に対応する文字を、*stream* 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。

ファイル中の st_ctime と st_mtime の両フィールドは、fputwc( ) 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。

putwc( )　putwc( ) 関数は、マクロとして実装される点を除いては、fputwc( ) 関数と同じです。

putwchar( )　putwchar(*wc*) の呼び出しは、putwc(*wc*, stdout) と同じです。putwchar( ) ルーチンはマクロとして実装されます。

戻り値　正常終了時、fputwc( )、putwc( )、および putwchar( ) は *wc* 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ー中のデータを書き出す必要があるとき、fputwc( )、putwc( )、および putwchar( ) 関数は以下の条件で異常終了します。

EAGAIN　*stream* 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　*stream* 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 *stream* 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
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 シグナルも呼び出し元スレッドへ送られる

fputwc()、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 fputwc() 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
インタフェースの安定性	標準

関連項目

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

名前

fputwc, putwc, putwchar – ワイド文字コードをストリームへ書き出す

形式

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

wint_t fputwc(wchar_t wc, FILE*stream);

wint_t putwc(wchar_t wc, FILE*stream);

#include <wchar.h>

wint_t putwchar(wchar_t wc);
```

機能説明

fputwc()

fputwc() 関数は、ワイド文字コード *wc* に対応する文字を、*stream* 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。

ファイル中の st_ctime と st_mtime の両フィールドは、fputwc() 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。

putwc()

putwc() 関数は、マクロとして実装される点を除いては、fputwc() 関数と同じです。

putwchar()

putwchar(*wc*) の呼び出しは、putwc(*wc*, stdout) と同じです。putwchar() ルーチンはマクロとして実装されます。

戻り値

正常終了時、fputwc()、putwc()、および putwchar() は *wc* 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ー中のデータを書き出す必要があるとき、fputwc()、putwc()、および putwchar() 関数は以下の条件で異常終了します。

EAGAIN — *stream* 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 — *stream* 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 *stream* 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
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 シグナルも呼び出し元スレッドへ送られる

fputwc()、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 fputwc() 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
インタフェースの安定性	標準

関連項目

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

名前

putws – ワイド文字列の EUC 文字への変換

形式

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

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 *restrict format, ...);

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

int sscanf(const char *restrict s, const char *restrict 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、初期シフトの状態で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)
- 長さ修飾子 (任意)。受け取るオブジェクトのサイズを指定します。
- 適用される変換形式を指定する変換指定文字 (有効な変換指定文字については、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 (入力が不適当) 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、有効な入力が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。入力項目の長さは、指定した最大フィールド幅によって制限されます。同時に、入力項目の長さの単位は、指定した変換文字によって文字またはバイト数のどちらかに解釈されます。scanf() は、入力項目の終わりを判断するために、余分な文字を読み取る必要がある場合があります。デフォルトの Solaris モードでは、入力項目

は「マッチングシーケンスの一部である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、複数の文字を読み取る必要があります。C99/SUSv3 モードでは、入力項目は「マッチングシーケンスと同じ (あるいは、その前半部分) である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、多くても 1 文字を読み取るだけでかまいません。したがって、C99/SUSv3 モードでは、strtod(3C) や strtol(3C) などの関数には受け入れられるシーケンスが scanf() では受け入れられない場合があります。どちらのモードでも、scanf() は、ungetc(3C) を使用して、超過バイトの読み取りをプッシュバックしようとします。このような試みがすべて成功したと想定した場合、入力項目の後に続く最初のバイト (もしあれば) は、読み取られずに残ります。入力項目の長さが 0 の場合、変換は失敗します。この状態はマッチングエラーです。ただし、ファイルの終わり、コード化エラー、または、読み取りエラーによって、ストリームからの入力が妨害された場合は入力エラーです。

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、変換文字に適当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当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。

長さ修飾子

次に、長さ修飾子とその意味を示します。

hh	後続の変換文字 d、i、o、u、x、X または n が signed char あるいは unsigned char へのポインタ型を持つ引数に適用されることを指定します。
h	後続の変換文字 d、i、o、u、x、X、または n が short あるいは unsigned short へのポインタ型を持つ引数に適用されることを指定します。
l (小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long あるいは unsigned long へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 a、A、e、E、f、F、g、または G が double へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 c、s、または [が wchar_t へのポインタ型を持つ引数に適用されることを指定します。
ll (小文字のエルと小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long long あるいは unsigned long long へのポインタ型を持つ引数に適用されることを指定します。

j　後続の変換文字 d、i、o、u、x、X、または n が intmax_t あるいは uintmax_t へのポインタ型を持つ引数に適用されることを指定します。

z　後続の変換文字 d、i、o、u、x、X、または n が size_t あるいは対応する符号付き整数型へのポインタ型を持つ引数に適用されることを指定します。

t　後続の変換文字 d、i、o、u、x、X、または n が ptrdiff_t あるいは対応する unsigned 型へのポインタ型を持つ引数に適用されることを指定します。

L　後続の変換文字 a、A、e、E、f、F、g、または G が long double へのポインタ型を持つ引数に適用されることを指定します。

長さ修飾子を上記以外の変換文字と一緒に指定した場合、その動作は未定義です。

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、strtol(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、strtol() 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。

a,e,f,g　符号付き浮動小数点数、無限、または NaN 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。変換文字 e、f、および g は、C99/SUSv3 モード (standards(5)

を参照) の場合だけ、16 進数の浮動小数点値に一致します。しかし、変換文字 a は常に、16 進数の浮動小数点値に一致します。

これらの変換文字は、strtod(3C) が受け入れる変換対象シーケンス (INF、INFINITY、NAN、および NAN (*n-char-sequence*) の形式を含む) にマッチングします。変換の結果は、マッチングシーケンスとともに strtod() (あるいは、strtof() または strtold()) を呼び出したときと同じで、浮動小数点の例外が発生して、(可能であれば)、errno に ERANGE が設定されます。

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。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 A、E、F、G、および X はそれぞれ a、e、f、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等です。

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 呼び出し

```
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 呼び出し

```
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
```

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
インタフェースの安定性	標準

関連項目

fgetc(3C), fgets(3C), fgetwc(3C), fread(3C), isspace(3C), printf(3C), setlocale(3C), strtod(3C), strtol(3C), strtoul(3C), wcrtomb(3C), ungetc(3C), attributes(5), standards(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* 引数には、次の値が指定できます。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 mbtowc(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。

LC_COLLATE は、 strcoll(3C) と 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 gettxt(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、

```
setlocale(LC_ALL, "C")
```

が実行されます。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。

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

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_TIME:	LC_ALL	LC_TIME	LANG
LC_NUMERIC:	LC_ALL	LC_NUMERIC:	LANG
LC_MONETARY:	LC_ALL	LC_MONETARY:	LANG
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

定義された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
CSI	対応済み
インタフェースの安定性	標準
MT レベル	例外付き MT-Safe

関連項目

locale(1), ctype(3C), getdate(3C) gettext(3C), gettxt(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), standards(5)

注意事項

マルチスレッドアプリケーションにおいて、あるスレッドがロケールに依存するルーチンを使用しているときに、ほかのスレッドがロケールを変更する (つまり、NULL 以外のロケール引数を指定して setlocale() を呼び出す) ことは危険です。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() を使用することは他のスレッドがロケールを変更していない限り可能で、マルチスレッドアプリケーションのどこで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。日本の時間表示は C ロケールの ASCII コードセットで表示できないからです。

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

形式

```
#include <stdio.h>

int printf(const char *restrict format, /* args*/ ...);

int fprintf(FILE *restrict stream, const char *restrict format, /* args*/ ...);

int sprintf(char *restrict s, const char *restrict format, /* args*/ ...);

int snprintf(char *restrict s, size_t n, const char *restrict 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* を追加したのと同等です。*n* には *s* によって参照されるバッファーのサイズが指定されます。*n* が 0 の場合、配列には何も書き込まれず、*s* はヌルポインタになる可能性もあります。そうでない場合、*n*-1 番目までの出力バイトが配列に書き込まれ、*n* 番目以降の出力バイトが破棄されます。そして、実際に配列に書き込まれたバイトの終わりにヌルバイトが書き込まれます。

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、使われません。

変換指定の書式 % の代わりに書式 %*n*$ を使用すると、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。

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。

  変換指定文字が 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。a、A、e、E、f 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指定文字で表される場合、動作は未定義です。

  変換指定文字が 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す。

  %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、その後に 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。

- 長さ修飾子 (任意)。引数のサイズを指定します。
- 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\
", 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`、`%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。 `a`、`A`、`e`、`E`、`f`、`F`、`g`、および `G` 変換の場合、必ず、結果には小数点が付きます。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, a, A, e, E, f, 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。

長さ修飾子

次に、長さ修飾子とその意味を示します。

`hh` 後続の変換文字 d、i、o、u、x、および X が `signed char` 型または `unsigned char` 型の引数に適用されることを指定します (引数は整数昇格の規則に従って昇格されますが、値は `signed char` または `unsigned char` の形式に変換されてから表示されます)。あるいは、後続の変換文字 n が `signed char` 型の引数へのポインタに適用されることを指定します。

`h` 後続の変換文字 d、i、o、u、x、および X が `short` 型または `unsigned short` 型の引数に適用されることを指定します (引数は整数昇格の規則に従って昇格されますが、値は `short` または `unsigned short` の形式に変換されてから表示されます)。あるいは、後続の変換文字 n が `short` 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) 後続の変換文字 d、i、o、u、x、または X が `long` 型あるいは `unsigned long` 型の引数に適用されることを指定します。あるいは、後続の変換文字 n が `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。あるいは、後続の変換文字 c が `wint_t` 型の引数に適用されることを指定します。あるいは、後続の変換文字 s が `wchar_t` 型の引数へのポインタに適用されることを指定します。あるいは、後続の変換文字 a、A、e、E、f、F、g、または G には効果がないことを指定します。

`ll` (小文字のエルと小文字のエル) 後続の変換文字 d、i、o、u、x、または X が `long long` 型あるいは `unsigned long long` 型の引数に適用されることを指定します。あるいは、後続の変換文字 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。

`j` 後続の変換文字 d、i、o、u、x、または X が `intmax_t` 型あるいは `uintmax_t` 型の引数に適用されることを指定します。あるいは、後続の変換文字 n が `intmax_t` 型の引数へのポインタに適用されることを指定します。

z　後続の変換文字 d、i、o、u、x または X が size_t 型あるいは対応する符号付き整数型の引数に適用されることを指定します。あるいは、後続の変換文字 n が size_t 型の引数に対応する符号付き整数型の引数へのポインタに適用されることを指定します。

t　後続の変換文字 d、i、o、u、x または X が ptrdiff_t 型あるいは対応する符号なし整数型の引数に適用されることを指定します。あるいは、後続の変換文字 n が ptrdiff_t 型の引数へのポインタに適用されることを指定します。

L　後続の変換文字 a、A、e、E、f、F、g、または G が long double 型の引数に適用されることを指定します。

長さ修飾子を上記以外の変換文字と一緒に指定した場合、その動作は未定義です。

変換指定文字　各変換指定文字を指定すると、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。

変換指定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 変換指定文字と同じ動作です。

f, F　　double の引数は、[–]*ddd*.*ddd* の形式で 10 進数に変換されます。小数点文字 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 に指定し、# フラグを指定しない場合、小数点文字は出力されません。小数点文字が出力されると、1 個以上の数字がその前に付きます。変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。

変換文字 f の場合、無限または非数を表す double 型の引数は、変換文字 e の形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “infinity” または “Infinity” と表示され、そうでない場合は “inf” または “Inf” と表示されます。

変換文字 F の場合、無限または非数を表す double 型の引数は、変換文字 E の SUSv3 形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “INFINITY” と表示され、そうでない場合は “INF” と表示されます。

e,E　　double の引数は、[–]*d*.*ddd*e±*dd* の形式に変換されます。小数点文字の前には 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)。小数点文字の後には、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、小数点文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。変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。値は適切な桁数に丸められます。

無限または非数の値は、次のように処理されます。

SUSv3　　変換文字 e の場合、無限を表す double 型の引数は、変換の精度が 7 以上の場合は “]&–]infinity” と表示され、そうでない場合は「[–]inf」と表示されます。非数を表す double 型の引数は、「[–]nan」と表示されます。変換文字 E の場合、「infinity」、「inf」、および「nan」の代わりに、それぞれ、「INFINITY」、「INF」、および「NAN」と表示されます。符号の印刷は、上記の規則に従います。

デフォルト　　無限を表す double 型の引数は、変換の精度が 7 以上の場合は “]&–]Infinity” と表示され、そうでない場合は “]&–]Inf” と表示されます。非数を表す double 型の引数は、“]&–]NaN” と表示されます。符号の印刷は、上記の規則に従います。

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。小数点文字は、その後に数字が続く場合だけ出力されます。

無限または非数を表す double 型の引数は、変換文字 e または E の形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “infinity”、“INFINITY”、または “Infinity” と表示され、そうでない場合は “inf”、“INF”、または “Inf” と表示されます。

a, A　　浮動小数点数を表す double 型の引数は、[-]0*xh.hhhhp±d* の形式に変換されます。基数点の前にある 1 桁の 16 進数 (h) は、変換された値が 0 の場合は 0 になり、そうでない場合は 1 になります。基数点の後にある 16 進数 (hhhh) の桁数は精度と同じになります。精度を指定しなかった場合、基数点の後にある 16 進数の桁数は、double 型の値を変換するときは 13 になり、long double 型の値を変換するときは 16 (x86 の場合) または 28 (SPARC の場合) になります。精度に 0 を指定し、# フラグを指定しなかった場合、10 進小数点文字は表示されません。変換文字 a の場合は文字「abcdef」が使用され、変換文字 A の場合は文字「ABCDEF」が使用されます。変換文字 A を指定した場合、「x」と「p」の代わりに、それぞれ、「X」と「P」が表示されます。指数の桁数は必ず 1 桁が表示され、2 の 10 進指数を表すのに必要な桁まで表示されます。値が 0 の場合、指数は 0 になります。

変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。

無限または非数を表す double 型の引数は、変換文字 e または E の SUSv3 形式に変換されます。

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、精度は付けないで、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、wcrtomb(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、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。

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() 関数は *n* が十分な大きさであったと仮定した場合に *s* に書き込まれる (はずの) バイト数を、末尾のヌルバイト (の分) を含まない値で返します。 snprintf() への呼び出しで *n* の値が 0 の場合、書き込まれるバイト数を返し、このとき *s* はヌルポインタとすることができます。

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。

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。

\　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

\\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\
"
```

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\
"
```

生成されるメッセージは次のとおりです。

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

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。 (続き)

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	下記参照
CSI	対応済み
インタフェースの安定性	標準

sprintf() 関数と snprintf() 関数は、「非同期シグナル安全」です。printf() 関数と fprintf() 関数は、setlocale(3C) を呼び出してロケールを変更していない限り、マルチスレッドアプリケーションで安全に使用できます。

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), vprintf(3C), wcstombs(3C), wctomb(3C), attributes(5), environ(5), standards(5)

注意事項

Solaris 10 より前のリリースで c89 を使用してコンパイルした 32 ビットアプリケーションでは、長さ修飾子 j を使用した場合の動作は未定義です。

n = 0 のときに snprintf() が返す値は、Solaris 10 リリースで変更されました。この変更は、SUSv3 仕様に基づいています。以前の動作は初期の SUSv2 仕様に基づいており、*n* = 0 のときに snprintf() が返す値は 1 よりも小さな未指定の値でした。

名前

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

形式

```
#include <stdio.h>

int printf(const char *restrict format, /* args*/ ...);

int fprintf(FILE *restrict stream, const char *restrict format, /* args*/ ...);

int sprintf(char *restrict s, const char *restrict format, /* args*/ ...);

int snprintf(char *restrict s, size_t n, const char *restrict 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* を追加したのと同等です。*n* には *s* によって参照されるバッファーのサイズが指定されます。*n* が 0 の場合、配列には何も書き込まれず、*s* はヌルポインタになる可能性もあります。そうでない場合、*n*-1 番目までの出力バイトが配列に書き込まれ、*n* 番目以降の出力バイトが破棄されます。そして、実際に配列に書き込まれたバイトの終わりにヌルバイトが書き込まれます。

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、使われません。

変換指定の書式 % の代わりに書式 %*n*$ を使用すると、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。

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。

  変換指定文字が 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。a、A、e、E、f 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指定文字で表される場合、動作は未定義です。

  変換指定文字が 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す。

  %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、その後に 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。

- 長さ修飾子 (任意)。引数のサイズを指定します。
- 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\
", 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、%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。 a、A、e、E、f、F、g、および G 変換の場合、必ず、結果には小数点が付きます。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, a, A, e, E, f, 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。

長さ修飾子

次に、長さ修飾子とその意味を示します。

hh 後続の変換文字 d、i、o、u、x、および X が signed char 型または unsigned char 型の引数に適用されることを指定します (引数は整数昇格の規則に従って昇格されますが、値は signed char または unsigned char の形式に変換されてから表示されます)。あるいは、後続の変換文字 n が signed char 型の引数へのポインタに適用されることを指定します。

h 後続の変換文字 d、i、o、u、x、および X が short 型または unsigned short 型の引数に適用されることを指定します (引数は整数昇格の規則に従って昇格されますが、値は short または unsigned short の形式に変換されてから表示されます)。あるいは、後続の変換文字 n が short 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) 後続の変換文字 d、i、o、u、x、または X が long 型あるいは unsigned long 型の引数に適用されることを指定します。あるいは、後続の変換文字 n が 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。あるいは、後続の変換文字 c が wint_t 型の引数に適用されることを指定します。あるいは、後続の変換文字 s が wchar_t 型の引数へのポインタに適用されることを指定します。あるいは、後続の変換文字 a、A、e、E、f、F、g、または G には効果がないことを指定します。

ll (小文字のエルと小文字のエル) 後続の変換文字 d、i、o、u、x、または X が long long 型あるいは unsigned long long 型の引数に適用されることを指定します。あるいは、後続の変換文字 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。

j 後続の変換文字 d、i、o、u、x、または X が intmax_t 型あるいは uintmax_t 型の引数に適用されることを指定します。あるいは、後続の変換文字 n が intmax_t 型の引数へのポインタに適用されることを指定します。

z　後続の変換文字 d、i、o、u、x または X が size_t 型あるいは対応する符号付き整数型の引数に適用されることを指定します。あるいは、後続の変換文字 n が size_t 型の引数に対応する符号付き整数型の引数へのポインタに適用されることを指定します。

t　後続の変換文字 d、i、o、u、x または X が ptrdiff_t 型あるいは対応する符号なし整数型の引数に適用されることを指定します。あるいは、後続の変換文字 n が ptrdiff_t 型の引数へのポインタに適用されることを指定します。

L　後続の変換文字 a、A、e、E、f、F、g、または G が long double 型の引数に適用されることを指定します。

長さ修飾子を上記以外の変換文字と一緒に指定した場合、その動作は未定義です。

変換指定文字

各変換指定文字を指定すると、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。

変換指定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 変換指定文字と同じ動作です。

f, F　double の引数は、[–]*ddd*.*ddd* の形式で 10 進数に変換されます。小数点文字 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 に指定し、# フラグを指定しない場合、小数点文字は出力されません。小数点文字が出力されると、1 個以上の数字がその前に付きます。変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。

変換文字 f の場合、無限または非数を表す double 型の引数は、変換文字 e の形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “infinity” または “Infinity” と表示され、そうでない場合は “inf” または “Inf” と表示されます。

変換文字 F の場合、無限または非数を表す double 型の引数は、変換文字 E の SUSv3 形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “INFINITY” と表示され、そうでない場合は “INF” と表示されます。

e,E　double の引数は、[–]*d*.*ddd*e±*dd* の形式に変換されます。小数点文字の前には 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)。小数点文字の後には、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、小数点文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。変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。値は適切な桁数に丸められます。

無限または非数の値は、次のように処理されます。

SUSv3　変換文字 e の場合、無限を表す double 型の引数は、変換の精度が 7 以上の場合は “]&–]infinity” と表示され、そうでない場合は「[–]inf」と表示されます。非数を表す double 型の引数は、「[–]nan」と表示されます。変換文字 E の場合、「infinity」、「inf」、および「nan」の代わりに、それぞれ、「INFINITY」、「INF」、および「NAN」と表示されます。符号の印刷は、上記の規則に従います。

デフォルト　無限を表す double 型の引数は、変換の精度が 7 以上の場合は “]&–]Infinity” と表示され、そうでない場合は “]&–]Inf” と表示されます。非数を表す double 型の引数は、“]&–]NaN” と表示されます。符号の印刷は、上記の規則に従います。

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。小数点文字は、その後に数字が続く場合だけ出力されます。

無限または非数を表す double 型の引数は、変換文字 e または E の形式に変換されます。ただし、引数が無限の場合、変換の精度が 8 以上の場合は “infinity”、“INFINITY”、または “Infinity” と表示され、そうでない場合は “inf”、“INF”、または “Inf” と表示されます。

a, A　　浮動小数点数を表す double 型の引数は、[-]0*xh.hhhhp±d* の形式に変換されます。基数点の前にある 1 桁の 16 進数 (h) は、変換された値が 0 の場合は 0 になり、そうでない場合は 1 になります。基数点の後にある 16 進数 (hhhh) の桁数は精度と同じになります。精度を指定しなかった場合、基数点の後にある 16 進数の桁数は、double 型の値を変換するときは 13 になり、long double 型の値を変換するときは 16 (x86 の場合) または 28 (SPARC の場合) になります。精度に 0 を指定し、# フラグを指定しなかった場合、10 進小数点文字は表示されません。変換文字 a の場合は文字「abcdef」が使用され、変換文字 A の場合は文字「ABCDEF」が使用されます。変換文字 A を指定した場合、「x」と「p」の代わりに、それぞれ、「X」と「P」が表示されます。指数の桁数は必ず 1 桁が表示され、2 の 10 進指数を表すのに必要な桁まで表示されます。値が 0 の場合、指数は 0 になります。

変換された値は、一般的な浮動小数点丸め方向モードに従って、指定した出力形式に合うように丸められます。変換が正確でない場合、不正確な例外が発生します。

無限または非数を表す double 型の引数は、変換文字 e または E の SUSv3 形式に変換されます。

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、精度は付けないで、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、wcrtomb(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、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。

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() 関数は *n* が十分な大きさであったと仮定した場合に *s* に書き込まれる (はずの) バイト数を、末尾のヌルバイト (の分) を含まない値で返します。 snprintf() への呼び出しで *n* の値が 0 の場合、書き込まれるバイト数を返し、このとき *s* はヌルポインタとすることができます。

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。

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。

\　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

\\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\
"
```

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\
"
```

生成されるメッセージは次のとおりです。

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

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。 *(続き)*

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	下記参照
CSI	対応済み
インタフェースの安定性	標準

sprintf() 関数と snprintf() 関数は、「非同期シグナル安全」です。printf() 関数と fprintf() 関数は、setlocale(3C) を呼び出してロケールを変更していない限り、マルチスレッドアプリケーションで安全に使用できます。

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), vprintf(3C), wcstombs(3C), wctomb(3C), attributes(5), environ(5), standards(5)

注意事項

Solaris 10 より前のリリースで c89 を使用してコンパイルした 32 ビットアプリケーションでは、長さ修飾子 j を使用した場合の動作は未定義です。

$n = 0$ のときに snprintf() が返す値は、Solaris 10 リリースで変更されました。この変更は、SUSv3 仕様に基づいています。以前の動作は初期の SUSv2 仕様に基づいており、$n = 0$ のときに snprintf() が返す値は 1 よりも小さな未指定の値でした。

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 *restrict format, ...);

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

int sscanf(const char *restrict s, const char *restrict 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、初期シフトの状態で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)
- 長さ修飾子 (任意)。受け取るオブジェクトのサイズを指定します。
- 適用される変換形式を指定する変換指定文字 (有効な変換指定文字については、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 (入力が不適当) 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、有効な入力が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。入力項目の長さは、指定した最大フィールド幅によって制限されます。同時に、入力項目の長さの単位は、指定した変換文字によって文字またはバイト数のどちらかに解釈されます。scanf() は、入力項目の終わりを判断するために、余分な文字を読み取る必要がある場合があります。デフォルトの Solaris モードでは、入力項目

は「マッチングシーケンスの一部である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、複数の文字を読み取る必要があります。C99/SUSv3 モードでは、入力項目は「マッチングシーケンスと同じ (あるいは、その前半部分) である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、多くても 1 文字を読み取るだけでかまいません。したがって、C99/SUSv3 モードでは、strtod(3C) や strtol(3C) などの関数には受け入れられるシーケンスが scanf() では受け入れられない場合があります。どちらのモードでも、scanf() は、ungetc(3C) を使用して、超過バイトの読み取りをプッシュバックしようとします。このような試みがすべて成功したと想定した場合、入力項目の後に続く最初のバイト (もしあれば) は、読み取られずに残ります。入力項目の長さが 0 の場合、変換は失敗します。この状態はマッチングエラーです。ただし、ファイルの終わり、コード化エラー、または、読み取りエラーによって、ストリームからの入力が妨害された場合は入力エラーです。

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、変換文字に適当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当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。

長さ修飾子

次に、長さ修飾子とその意味を示します。

hh	後続の変換文字 d、i、o、u、x、X または n が signed char あるいは unsigned char へのポインタ型を持つ引数に適用されることを指定します。
h	後続の変換文字 d、i、o、u、x、X、または n が short あるいは unsigned short へのポインタ型を持つ引数に適用されることを指定します。
l (小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long あるいは unsigned long へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 a、A、e、E、f、F、g、または G が double へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 c、s、または [が wchar_t へのポインタ型を持つ引数に適用されることを指定します。
ll (小文字のエルと小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long long あるいは unsigned long long へのポインタ型を持つ引数に適用されることを指定します。

j　後続の変換文字 d、i、o、u、x、X、または n が intmax_t あるいは uintmax_t へのポインタ型を持つ引数に適用されることを指定します。

z　後続の変換文字 d、i、o、u、x、X、または n が size_t あるいは対応する符号付き整数型へのポインタ型を持つ引数に適用されることを指定します。

t　後続の変換文字 d、i、o、u、x、X、または n が ptrdiff_t あるいは対応する unsigned 型へのポインタ型を持つ引数に適用されることを指定します。

L　後続の変換文字 a、A、e、E、f、F、g、または G が long double へのポインタ型を持つ引数に適用されることを指定します。

長さ修飾子を上記以外の変換文字と一緒に指定した場合、その動作は未定義です。

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、strtol(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、strtol() 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。

a,e,f,g　符号付き浮動小数点数、無限、または NaN 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。変換文字 e、f、および g は、C99/SUSv3 モード (standards(5)

を参照) の場合だけ、16 進数の浮動小数点値に一致します。しかし、変換文字 a は常に、16 進数の浮動小数点値に一致します。

これらの変換文字は、strtod(3C) が受け入れる変換対象シーケンス (INF、INFINITY、NAN、および NAN (*n-char-sequence*) の形式を含む) にマッチングします。変換の結果は、マッチングシーケンスとともに strtod() (あるいは、strtof() または strtold()) を呼び出したときと同じで、浮動小数点の例外が発生して、(可能であれば)、errno に ERANGE が設定されます。

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。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 A、E、F、G、および X はそれぞれ a、e、f、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等です。

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 呼び出し

```
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 呼び出し

```
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
```

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
インタフェースの安定性	標準

関連項目

`fgetc`(3C), `fgets`(3C), `fgetwc`(3C), `fread`(3C), `isspace`(3C), `printf`(3C), `setlocale`(3C), `strtod`(3C), `strtol`(3C), `strtoul`(3C), `wcrtomb`(3C), `ungetc`(3C), `attributes`(5), `standards`(5)

名前

strftime, cftime, ascftime – 日付と時刻を文字列に変換

形式

```
#include <time.h>

size_t strftime(char *restrict s, size_t maxsize, const char *restrict format,
     const struct tm *restrict timeptr);

int cftime(char *s, char *format, const time_t *clock);

int ascftime(char *s, const char *format, const struct tm *timeptr);
```

機能説明

strftime()、ascftime()、cftime() 関数は、*format* がポイントする文字列の制御に従って、バイトを *s* がポイントする配列に格納します。*format* 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 *s* が示す配列に複写されます。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。strftime() の場合は、*maxsize* を超えないバイト数の文字だけが配列に格納されます。

format 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すが、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。

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 *timeptr* がポイントする構造体に入っている値、cftime() の *clock* で表されている時間で決まります。

%% % 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)

デフォルト

%c 日時の表現 (ロケール固有)

```
%a %b %d %H:%M:%S %Y
```

これは、Solaris 2.4 より以前に最初にサポートされた標準準拠と同様に、デフォルトの表現です。 standards(5) を参照してください。

標準準拠

%c 日時の表現 (ロケール固有)

```
%a %b %e %H:%M:%S %Y
```

これは、Solaris 2.4 で最初にサポートされて以来、Solaris 10 にいたるまでの標準準拠の表現です。

デフォルト　%C　date(1) が出力する形式の日時の表現

これは、Solaris 2.4 より以前に最初にサポートされた標準準拠と同様に、デフォルトの表現です。

標準準拠　%C　世紀を 01 から 99 までの数値で表現 (年を 100 で割って整数に切り捨て)

これは、Solaris 2.4 で最初にサポートされて以来、Solaris 10 にいたるまでの標準準拠の表現です。

%d　日 (01 から 31)。

%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

%F　%Y-%m-%d (ISO 8601:2000 標準の日付形式) と同じ

%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 (00 から 23)。

%I　12 時間表示で表した時 (01 から 12)。

%j　年の通算日 (001 から 366)。

%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 (01 から 12)。

%M　分 (00 から 59)。

%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 60)。値の範囲が (00 から 59) ではなく、(00 から 60) であるのは、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)。月曜日が 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　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 (00 から 99)

%Y　西暦年の完全表示 (たとえば 1993)

%z　ISO 8601:2000 標準形式 (+hhmm または -hhmm) の UTC からのオフセットに置き換わります。タイムゾーンがわからない場合には、文字列には置き換えられません。たとえば、"-0430" は、UTC から 4 時間 30 分おくれ (グリニッジから西に離れていること) を意味します。tm_isdst が 0 の場合には、標準時間のオフセットが使用されます。tm_isdst が正の値の場合には、夏時間のオフセットが使用されます。tm_isdst が負の値の場合には、文字列は返されません。

%Z　タイムゾーン名または省略形 (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
%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、strftime()、cftime()、ascftime() は米国英語で表示されます。setlocale() で LC_TIME カテゴリのロケールを設定すると、strftime()、cftime()、ascftime() の実行結果を特定の言語で出力できるようになります。

時間帯　tzset(3C) 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

`strftime()`、`cftime()`、`ascftime()` 関数は、*s* 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 NULL 文字は含まれません。生成された文字の合計数 (終了 NULL 文字を含む) が *maxsize* を超えていると、`strftime()` は、0 を返し、配列の内容は未確定になります。

使用例

例 1 `strftime()` 関数の例

ここでは、POSIX ロケールの `strftime()` の使用例を示します。*tmptr* 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 `str` の内容がどうなるかを示します。

```
strftime (str, strsize, "%A %b %d %j", tmptr)
```

これにより、`str` の内容は `"Thursday Aug 28 240"` になります。

属性

以下の属性については、`attributes`(5) を参照してください。

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

関連項目

`date`(1), `ctime`(3C), `mktime`(3C), `setlocale(3C)`, `strptime`(3C), `tzset`(3C), `TIMEZONE`(4), `zoneinfo`(4), `attributes`(5), `environ`(5), `standards`(5)

注意事項

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、新しい年の日数が 4 日未満含まれている場合、その週は前年の第 53 週となっていました。しかし、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。

`%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。これらの仕様は変更される可能性があります。ISO C9x 標準がこれらの仕様と異なる決定をした場合、仕様は ISO C9x 標準の決定に準拠して変更されます。

Solaris 8 リリースでは、`%u` の変換仕様が変更されました。この変更は、XPG4 仕様に基づいたものです。

`%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

```
#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);

#include <libintl.h>
#include <locale.h>

char *dcgettext(const char *domainname, const char *msgid, int category);
```

GNU 互換

```
#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。*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。

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 *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)

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
インタフェースの安定性	標準

関連項目　isascii(3C), `attributes`(5), `standards`(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()` 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
インタフェースの安定性	標準

関連項目　`isupper(3C)`, `tolower(3C)`, `attributes(5)`, `standards(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、 引数が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
インタフェースの安定性	標準

関連項目

_tolower(3C), setlocale(3C), attributes(5), standards(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() 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
インタフェースの安定性	標準

関連項目　islower(3C), toupper(3C), attributes(5), standards(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
インタフェースの安定性	標準

関連項目　_toupper(3C), setlocale(3C), attributes(5), standards(5)

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
インタフェースの安定性	標準

関連項目

iswalpha(3C), setlocale(3C), towupper(3C), attributes(5), standards(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
インタフェースの安定性	標準

関連項目

iswalpha(3C), setlocale(3C), towlower(3C), attributes(5), standards(5)

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、途中に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いことがあります。

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
インタフェースの安定性	標準

関連項目

read(2), fseek(3C), fsetpos(3C), rewind(3C), setbuf(3C), attributes(5), standards(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 *restrict format, ...);

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

int sscanf(const char *restrict s, const char *restrict 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、初期シフトの状態で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)
- 長さ修飾子 (任意)。受け取るオブジェクトのサイズを指定します。
- 適用される変換形式を指定する変換指定文字 (有効な変換指定文字については、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 (入力が不適当) 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、有効な入力が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。入力項目の長さは、指定した最大フィールド幅によって制限されます。同時に、入力項目の長さの単位は、指定した変換文字によって文字またはバイト数のどちらかに解釈されます。scanf() は、入力項目の終わりを判断するために、余分な文字を読み取る必要がある場合があります。デフォルトの Solaris モードでは、入力項目

は「マッチングシーケンスの一部である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、複数の文字を読み取る必要があります。C99/SUSv3 モードでは、入力項目は「マッチングシーケンスと同じ (あるいは、その前半部分) である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、多くても 1 文字を読み取るだけでかまいません。したがって、C99/SUSv3 モードでは、strtod(3C) や strtol(3C) などの関数には受け入れられるシーケンスが scanf() では受け入れられない場合があります。どちらのモードでも、scanf() は、ungetc(3C) を使用して、超過バイトの読み取りをプッシュバックしようとします。このような試みがすべて成功したと想定した場合、入力項目の後に続く最初のバイト (もしあれば) は、読み取られずに残ります。入力項目の長さが 0 の場合、変換は失敗します。この状態はマッチングエラーです。ただし、ファイルの終わり、コード化エラー、または、読み取りエラーによって、ストリームからの入力が妨害された場合は入力エラーです。

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、変換文字に適当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当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。

長さ修飾子

次に、長さ修飾子とその意味を示します。

hh	後続の変換文字 d、i、o、u、x、X または n が signed char あるいは unsigned char へのポインタ型を持つ引数に適用されることを指定します。
h	後続の変換文字 d、i、o、u、x、X、または n が short あるいは unsigned short へのポインタ型を持つ引数に適用されることを指定します。
l (小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long あるいは unsigned long へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 a、A、e、E、f、F、g、または G が double へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 c、s、または [が wchar_t へのポインタ型を持つ引数に適用されることを指定します。
ll (小文字のエルと小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long long あるいは unsigned long long へのポインタ型を持つ引数に適用されることを指定します。

j　後続の変換文字 d、i、o、u、x、X、または n が intmax_t あるいは uintmax_t へのポインタ型を持つ引数に適用されることを指定します。

z　後続の変換文字 d、i、o、u、x、X、または n が size_t あるいは対応する符号付き整数型へのポインタ型を持つ引数に適用されることを指定します。

t　後続の変換文字 d、i、o、u、x、X、または n が ptrdiff_t あるいは対応する unsigned 型へのポインタ型を持つ引数に適用されることを指定します。

L　後続の変換文字 a、A、e、E、f、F、g、または G が long double へのポインタ型を持つ引数に適用されることを指定します。

長さ修飾子を上記以外の変換文字と一緒に指定した場合、その動作は未定義です。

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、strtol(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、strtol() 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。

a,e,f,g　符号付き浮動小数点数、無限、または NaN 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。変換文字 e、f、および g は、C99/SUSv3 モード (standards(5)

を参照) の場合だけ、16 進数の浮動小数点値に一致します。しかし、変換文字 a は常に、16 進数の浮動小数点値に一致します。

これらの変換文字は、strtod(3C) が受け入れる変換対象シーケンス (INF、INFINITY、NAN、および NAN (*n-char-sequence*) の形式を含む) にマッチングします。変換の結果は、マッチングシーケンスとともに strtod() (あるいは、strtof() または strtold()) を呼び出したときと同じで、浮動小数点の例外が発生して、(可能であれば)、errno に ERANGE が設定されます。

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。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 A、E、F、G、および X はそれぞれ a、e、f、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等です。

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 呼び出し

```
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 呼び出し

```
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
```

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
インタフェースの安定性	標準

関連項目

`fgetc`(3C), `fgets`(3C), `fgetwc`(3C), `fread`(3C), `isspace`(3C), `printf`(3C), `setlocale`(3C), `strtod`(3C), `strtol`(3C), `strtoul`(3C), `wcrtomb`(3C), `ungetc`(3C), `attributes`(5), `standards`(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 *restrict format, ...);

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

int sscanf(const char *restrict s, const char *restrict 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、初期シフトの状態で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)
- 長さ修飾子 (任意)。受け取るオブジェクトのサイズを指定します。
- 適用される変換形式を指定する変換指定文字 (有効な変換指定文字については、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 (入力が不適当) 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、有効な入力が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。入力項目の長さは、指定した最大フィールド幅によって制限されます。同時に、入力項目の長さの単位は、指定した変換文字によって文字またはバイト数のどちらかに解釈されます。scanf() は、入力項目の終わりを判断するために、余分な文字を読み取る必要がある場合があります。デフォルトの Solaris モードでは、入力項目

は「マッチングシーケンスの一部である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、複数の文字を読み取る必要があります。C99/SUSv3 モードでは、入力項目は「マッチングシーケンスと同じ (あるいは、その前半部分) である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、多くても 1 文字を読み取るだけでかまいません。したがって、C99/SUSv3 モードでは、strtod(3C) や strtol(3C) などの関数には受け入れられるシーケンスが scanf() では受け入れられない場合があります。どちらのモードでも、scanf() は、ungetc(3C) を使用して、超過バイトの読み取りをプッシュバックしようとします。このような試みがすべて成功したと想定した場合、入力項目の後に続く最初のバイト (もしあれば) は、読み取られずに残ります。入力項目の長さが 0 の場合、変換は失敗します。この状態はマッチングエラーです。ただし、ファイルの終わり、コード化エラー、または、読み取りエラーによって、ストリームからの入力が妨害された場合は入力エラーです。

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、変換文字に適当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当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。

長さ修飾子

次に、長さ修飾子とその意味を示します。

hh	後続の変換文字 d、i、o、u、x、X または n が signed char あるいは unsigned char へのポインタ型を持つ引数に適用されることを指定します。
h	後続の変換文字 d、i、o、u、x、X、または n が short あるいは unsigned short へのポインタ型を持つ引数に適用されることを指定します。
l (小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long あるいは unsigned long へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 a、A、e、E、f、F、g、または G が double へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 c、s、または [が wchar_t へのポインタ型を持つ引数に適用されることを指定します。
ll (小文字のエルと小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long long あるいは unsigned long long へのポインタ型を持つ引数に適用されることを指定します。

j　後続の変換文字 d、i、o、u、x、X、または n が intmax_t あるいは uintmax_t へのポインタ型を持つ引数に適用されることを指定します。

z　後続の変換文字 d、i、o、u、x、X、または n が size_t あるいは対応する符号付き整数型へのポインタ型を持つ引数に適用されることを指定します。

t　後続の変換文字 d、i、o、u、x、X、または n が ptrdiff_t あるいは対応する unsigned 型へのポインタ型を持つ引数に適用されることを指定します。

L　後続の変換文字 a、A、e、E、f、F、g、または G が long double へのポインタ型を持つ引数に適用されることを指定します。

長さ修飾子を上記以外の変換文字と一緒に指定した場合、その動作は未定義です。

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、strtol(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、strtol() 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。

a,e,f,g　符号付き浮動小数点数、無限、または NaN 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。変換文字 e、f、および g は、C99/SUSv3 モード (standards(5)

を参照) の場合だけ、16 進数の浮動小数点値に一致します。しかし、変換文字 a は常に、16 進数の浮動小数点値に一致します。

これらの変換文字は、strtod(3C) が受け入れる変換対象シーケンス (INF、INFINITY、NAN、および NAN (*n-char-sequence*) の形式を含む) にマッチングします。変換の結果は、マッチングシーケンスとともに strtod() (あるいは、strtof() または strtold()) を呼び出したときと同じで、浮動小数点の例外が発生して、(可能であれば)、errno に ERANGE が設定されます。

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。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 A、E、F、G、および X はそれぞれ a、e、f、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等です。

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 呼び出し

```
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 呼び出し

```
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
```

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
インタフェースの安定性	標準

関連項目

fgetc(3C), fgets(3C), fgetwc(3C), fread(3C), isspace(3C), printf(3C), setlocale(3C), strtod(3C), strtol(3C), strtoul(3C), wcrtomb(3C), ungetc(3C), attributes(5), standards(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 *restrict format, ...);

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

int sscanf(const char *restrict s, const char *restrict 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、初期シフトの状態で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)
- 長さ修飾子 (任意)。受け取るオブジェクトのサイズを指定します。
- 適用される変換形式を指定する変換指定文字 (有効な変換指定文字については、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 (入力が不適当) 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、有効な入力が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。入力項目の長さは、指定した最大フィールド幅によって制限されます。同時に、入力項目の長さの単位は、指定した変換文字によって文字またはバイト数のどちらかに解釈されます。scanf() は、入力項目の終わりを判断するために、余分な文字を読み取る必要がある場合があります。デフォルトの Solaris モードでは、入力項目

は「マッチングシーケンスの一部である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、複数の文字を読み取る必要があります。C99/SUSv3 モードでは、入力項目は「マッチングシーケンスと同じ (あるいは、その前半部分) である入力バイトの最長のシーケンス」であると定義されているため、scanf() は、多くても 1 文字を読み取るだけでかまいません。したがって、C99/SUSv3 モードでは、strtod(3C) や strtol(3C) などの関数には受け入れられるシーケンスが scanf() では受け入れられない場合があります。どちらのモードでも、scanf() は、ungetc(3C) を使用して、超過バイトの読み取りをプッシュバックしようとします。このような試みがすべて成功したと想定した場合、入力項目の後に続く最初のバイト (もしあれば) は、読み取られずに残ります。入力項目の長さが 0 の場合、変換は失敗します。この状態はマッチングエラーです。ただし、ファイルの終わり、コード化エラー、または、読み取りエラーによって、ストリームからの入力が妨害された場合は入力エラーです。

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、変換文字に適当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当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。

長さ修飾子

次に、長さ修飾子とその意味を示します。

hh	後続の変換文字 d、i、o、u、x、X または n が signed char あるいは unsigned char へのポインタ型を持つ引数に適用されることを指定します。
h	後続の変換文字 d、i、o、u、x、X、または n が short あるいは unsigned short へのポインタ型を持つ引数に適用されることを指定します。
l (小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long あるいは unsigned long へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 a、A、e、E、f、F、g、または G が double へのポインタ型を持つ引数に適用されることを指定します。あるいは、後続の変換文字 c、s、または [が wchar_t へのポインタ型を持つ引数に適用されることを指定します。
ll (小文字のエルと小文字のエル)	後続の変換文字 d、i、o、u、x、X、または n が long long あるいは unsigned long long へのポインタ型を持つ引数に適用されることを指定します。

j　後続の変換文字 d、i、o、u、x、X、または n が intmax_t あるいは uintmax_t へのポインタ型を持つ引数に適用されることを指定します。

z　後続の変換文字 d、i、o、u、x、X、または n が size_t あるいは対応する符号付き整数型へのポインタ型を持つ引数に適用されることを指定します。

t　後続の変換文字 d、i、o、u、x、X、または n が ptrdiff_t あるいは対応する unsigned 型へのポインタ型を持つ引数に適用されることを指定します。

L　後続の変換文字 a、A、e、E、f、F、g、または G が long double へのポインタ型を持つ引数に適用されることを指定します。

長さ修飾子を上記以外の変換文字と一緒に指定した場合、その動作は未定義です。

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、strtol(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、strtol() 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。

a,e,f,g　符号付き浮動小数点数、無限、または NaN 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。変換文字 e、f、および g は、C99/SUSv3 モード (standards(5)

を参照) の場合だけ、16 進数の浮動小数点値に一致します。しかし、変換文字 a は常に、16 進数の浮動小数点値に一致します。

これらの変換文字は、strtod(3C) が受け入れる変換対象シーケンス (INF、INFINITY、NAN、および NAN (*n-char-sequence*) の形式を含む) にマッチングします。変換の結果は、マッチングシーケンスとともに strtod() (あるいは、strtof() または strtold()) を呼び出したときと同じで、浮動小数点の例外が発生して、(可能であれば)、errno に ERANGE が設定されます。

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。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 A、E、F、G、および X はそれぞれ a、e、f、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等です。

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 呼び出し

```
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 呼び出し

```
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
```

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
インタフェースの安定性	標準

関連項目

fgetc(3C), fgets(3C), fgetwc(3C), fread(3C), isspace(3C), printf(3C), setlocale(3C), strtod(3C), strtol(3C), strtoul(3C), wcrtomb(3C), ungetc(3C), attributes(5), standards(5)

名前

wcstod, wcstof, wcstold, wstod, watof – ワイド文字列を浮動小数点数に変換

形式

```
#include <wchar.h>

double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);

float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);

long double wcstold(const wchar_t *restrict nptr, wchar_t **restrict endptr);

double wstod(const wchar_t *nptr, wchar_t **endptr);

double watof(wchar_t *nptr);
```

機能説明

wcstod()、wcstof()、および wcstold() 関数は、*nptr* が指すワイド文字列の先頭部分を、それぞれ、double、float、および long double 型の表現に変換します。これらの関数はまず、指定されたワイド文字列を次の 3 つのシーケンスに分解します。

1. 最初のシーケンス。空白などのワイド文字から構成されます。つまり、iswspace(3C) で識別されるワイド文字です。このシーケンスは空の場合もあります。
2. 変換対象シーケンス。変換対象として認識されるワイド文字から構成されます。このシーケンスは、浮動小数点の定数として、あるいは、無限または非数を表す文字列として解釈されます。
3. 最後のシーケンス。変換対象として認識されないワイド文字から構成されます。つまり、終了ヌルなどのワイド文字です。このワイド文字は 1 文字の場合も複数文字の場合もあります。

次に、これらの関数は、変換対象シーケンスを浮動小数点数に変換して、その結果を返します。

変換対象シーケンスの期待される形式は、プラスまたはマイナス記号 (任意) の後に、次のいずれか 1 つが続くものです。

- 10 進数のシーケンス (空でない)。この後には、基数点 (任意) が続き、さらには、指数部 (任意) が続くこともあります。
- 0x または 0X とそれに続く 16 進数のシーケンス (空でない)。この後には、基数点 (任意) が続き、さらには、指数部 (任意) が続くこともあります。
- INF または INFINITY。あるいは、これと等価なワイド文字列 (大文字小文字の区別はない)。
- NAN または NAN (*n-wchar-sequence*$_{opt}$)。あるいは、これと等価なワイド文字列 (NAN 部分の大文字小文字の区別はない)。ここで、w-wchar-sequence は次のとおりです。

```
n-wchar-sequence:
    digit
    nondigit
    n-wchar-sequence digit
    n-wchar-sequence nondigit
```

wcstod() のデフォルトのモードでは、10 進数、INF/INFINITY、および、NAN/NAN (*n-char-sequence*) の形式だけが認識されます。C99/SUSv3 モードでは、これに加えて、16 進数の文字列も認識されます。

wcstod() のデフォルトのモードでは、NAN (*n-char-equence*) 形式の *n-char-sequence* 部分には、「)」(右の丸括弧) または「\」(ヌル) を除く、すべての文字を含むことができます。C99/SUSv3 モードでは、*n-char-sequence* には、英大文字、英小文字、数字、および「_」(下線) だけを含むことができます。

wcstof() 関数と wcstold() 関数は、常に、C99/SUSv3 準拠モードで動作します。

変換対象シーケンスは、「指定されたワイド文字列内で最初の空白以外のワイド文字から始まる最初で最長の部分シーケンス」であると定義されます。つまり、これを「期待される形式」と呼びます。変換対象シーケンスが期待される形式でない場合、変換対象シーケンスにワイド文字は含まれません。

変換対象シーケンスが浮動小数点数の期待される形式である場合、変換対象シーケンスは次のように解釈されます。変換対象シーケンスが数字または基数点文字から始まる場合、このシーケンスは、C 言語の規則に従って、浮動小数点の定数であると解釈されます。ただし、C 言語の規則と異なる点として、ピリオドの代わりに基数点文字が使用されます。また、変換対象シーケンスが 10 進数の浮動小数点数であり、かつ、指数部または基数点文字のどちらも存在しない場合、あるいは、変換対象シーケンスが 16 進数の浮動小数点数であり、かつ、2 進数の指数部が存在しない場合、変換対象シーケンスの最後にある数字の後に、値が 0 で適切な型の指数部が続くと想定されます。変換対象シーケンスがマイナス記号から始まる場合、このシーケンスは負であると解釈されます。変換対象シーケンスが INF または INIFITITY である場合、このシーケンスは無限であると解釈されます。変換対象シーケンスが NAN または NAN (*n-wchar-sequence*$_{opt}$) である場合、このシーケンスは非数であると解釈されます。*endptr* が指すオブジェクトには、最後のシーケンスへのポインタが格納されます。ただし、*endptr* がヌルポインタである場合を除きます。

変換対象シーケンスが 10 進数または 16 進数の形式である場合、変換の結果の値は、一般的な浮動小数点丸め方向モードに従って正しく丸められます。変換の結果、場合によっては、浮動小数点の不正確な例外、アンダーフローの例外、またはオーバーフローの例外が発生することもあります。

基数点文字は、プログラムのロケール (カテゴリ LC_NUMERIC) で定義されます。POSIX ロケールなど、基数点文字が定義されていないロケールでは、基数点文字のデフォルトはピリオド (.) です。

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。

wcstod() 関数は、正常終了した場合、errno の設定を変更しません。

wstod() 関数は、wcstod() と同等です。

watof(*str*) 関数は wstod(str, (wchar_t **)NULL) と同じです。

戻り値

これらの関数は、正常終了した場合、変換した値を返します。変換しなかった場合は、0 を返します。

変換された値が表現可能な値の範囲を超えてしまった場合、(値の符号によって)、±HUGE_VAL、±HUGE_VALF、または、±HUGE_VALL が返されます。そして、浮動小数点オーバーフローの例外が発生して、errno は ERANGE に設定されます。

値を変換した結果としてアンダーフローの例外が発生した場合、正しく丸められた結果 (通常、通常以下、または 0) が返されます。そして、浮動小数点アンダーフローの例外が発生して、errno は ERANGE 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()、wcstof()、wcstold() 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
インタフェースの安定性	wcstod()、wcstof()、および wcstold() は標準

関連項目

iswspace(3C), localeconv(3C), scanf(3C), setlocale(3C), wcstol(3C), attributes(5), standards(5)

名前

wcstol, wcstoll, wstol, watol, watoll, watoi – ワイド文字列を整数に変換

形式

```
#include <wchar.h>

long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr, int base);

long long wcstoll(const wchar_t *restrict nptr, wchar_t **restrict endptr, int
     base);

#include <widec.h>

long 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 `wcstoll()` の両関数は、*nptr* が示すワイド文字列の先頭部分を `long` および `long long` 型に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。

1. 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。
2. 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。
3. 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 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 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。

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。

これらの関数は、正常終了した場合、`errno` の設定を変更しません。

エラー時には、0、{`LONG_MIN`} または {`LLONG_MIN`}、および {`LONG_MAX`} または {`LLONG_MAX`} が返され、成功時には、有効な値が返されるため、エラー状態をチェックしたいアプリケーションは、これらの関数を呼び出す前に、`errno` を 0 に設定しておく必要があります。すると、後で `errno` をチェックできます。

`wstol()` 関数は、`wcstol()` と同等です。

`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、これらの関数は、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_MIN`}、{`LONG_MAX`}、{`LLONG_MIN`} または {`LLONG_MAX`} 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

これらの関数は、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

またこれらの関数は、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
インタフェースの安定性	wcstol() と wcstoll() は標準

関連項目

iswalpha(3C), iswspace(3C), scanf(3C), wcstod(3C), attributes(5), standards(5)

注意事項

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

wcstol, wcstoll, wstol, watol, watoll, watoi – ワイド文字列を整数に変換

形式

```
#include <wchar.h>

long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr, int base);

long long wcstoll(const wchar_t *restrict nptr, wchar_t **restrict endptr, int
     base);

#include <widec.h>

long 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 `wcstoll()` の両関数は、*nptr* が示すワイド文字列の先頭部分を `long` および `long long` 型に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。

1. 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。
2. 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。
3. 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 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 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。

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。

これらの関数は、正常終了した場合、`errno` の設定を変更しません。

エラー時には、0、{`LONG_MIN`} または {`LLONG_MIN`}、および {`LONG_MAX`} または {`LLONG_MAX`} が返され、成功時には、有効な値が返されるため、エラー状態をチェックしたいアプリケーションは、これらの関数を呼び出す前に、`errno` を 0 に設定しておく必要があります。すると、後で `errno` をチェックできます。

`wstol()` 関数は、`wcstol()` と同等です。

`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、これらの関数は、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_MIN`}、{`LONG_MAX`}、{`LLONG_MIN`} または {`LLONG_MAX`} 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

これらの関数は、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

またこれらの関数は、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
インタフェースの安定性	wcstol() と wcstoll() は標準

関連項目

iswalpha(3C), iswspace(3C), scanf(3C), wcstod(3C), attributes(5), standards(5)

注意事項

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

wcstol, wcstoll, wstol, watol, watoll, watoi – ワイド文字列を整数に変換

形式

```
#include <wchar.h>

long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr, int base);

long long wcstoll(const wchar_t *restrict nptr, wchar_t **restrict endptr, int
     base);

#include <widec.h>

long 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 `wcstoll()` の両関数は、*nptr* が示すワイド文字列の先頭部分を `long` および `long long` 型に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。

1. 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。
2. 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。
3. 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 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 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。

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。

これらの関数は、正常終了した場合、`errno` の設定を変更しません。

エラー時には、0、{`LONG_MIN`} または {`LLONG_MIN`}、および {`LONG_MAX`} または {`LLONG_MAX`} が返され、成功時には、有効な値が返されるため、エラー状態をチェックしたいアプリケーションは、これらの関数を呼び出す前に、`errno` を 0 に設定しておく必要があります。すると、後で `errno` をチェックできます。

`wstol()` 関数は、`wcstol()` と同等です。

`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、これらの関数は、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_MIN`}、{`LONG_MAX`}、{`LLONG_MIN`} または {`LLONG_MAX`} 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

これらの関数は、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

またこれらの関数は、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
インタフェースの安定性	wcstol() と wcstoll() は標準

関連項目

iswalpha(3C), iswspace(3C), scanf(3C), wcstod(3C), attributes(5), standards(5)

注意事項

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

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs( )` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn( )` と `wsspn( )` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn( )` と `wscspn( )` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok( )` と `wstok( )` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

`malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

注意事項

`wcscat()`、`wcsncat()`、`wcscmp()`、`wcsncmp()`、`wcscpy()`、`wcsncpy()`、`wcslen()`、`wcschr()`、`wcsrchr()`、`wcspbrk()`、`wcswcs()`、`wcsspn()`、`wcscspn()`、および `wcstok()` 関数は「標準」です。`wscat()`、`wsncat()`、`wscmp()`、`wscpy()`、`wsncpy()`、`wslen()`、`wschr()`、`wsrchr()`、`wspbrk()`、`wsspn()`、`wstok()`、`windex()`、および `wrindex()` 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

デフォルトとその他の標準仕様

```
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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcscoll, 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。

wcscoll() 関数と wscoll() 関数は、正常終了した場合、errno の設定を変更しません。

エラー状態をチェックしたいアプリケーションは、wcscoll() または wscoll() を呼び出す前に、errno を 0 に設定しておく必要があります。0 以外の errno が返った場合、エラーが発生していたことになります。

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

使用法

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
インタフェースの安定性	wcscoll() は標準

関連項目

setlocale(3C), wcscmp(3C), wcsxfrm(3C), attributes(5), standards(5)

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

デフォルトとその他の標準仕様

```
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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

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

これらの関数は、EUC ロケールだけに機能します。

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>
```

XPG4 と SUS

```
size_t wcsftime(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 wcsftime(wchar_t *restrict wcs, size_t maxsize, const wchar_t *restrict
     format, const struct tm *restrict timptr);
```

機能説明

wcsftime() 関数は、次の点を除いて strftime(3C) 関数と同じです。

- *wcs* 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
- *maxsize* 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
- *format* 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
- 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

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。

戻り値

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 *maxsize* 以下の場合、wcsftime() は、*wcs* 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 *maxsize* 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。

wcfstime() 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。

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
インタフェースの安定性	標準

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。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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かれば 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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つかれば 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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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かれば 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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つかれば 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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

`malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

注意事項

`wcscat()`、`wcsncat()`、`wcscmp()`、`wcsncmp()`、`wcscpy()`、`wcsncpy()`、`wcslen()`、`wcschr()`、`wcsrchr()`、`wcspbrk()`、`wcswcs()`、`wcsspn()`、`wcscspn()`、および `wcstok()` 関数は「標準」です。`wscat()`、`wsncat()`、`wscmp()`、`wscpy()`、`wsncpy()`、`wslen()`、`wschr()`、`wsrchr()`、`wspbrk()`、`wsspn()`、`wstok()`、`windex()`、および `wrindex()` 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

`malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

注意事項

`wcscat()`、`wcsncat()`、`wcscmp()`、`wcsncmp()`、`wcscpy()`、`wcsncpy()`、`wcslen()`、`wcschr()`、`wcsrchr()`、`wcspbrk()`、`wcswcs()`、`wcsspn()`、`wcscspn()`、および `wcstok()` 関数は「標準」です。`wscat()`、`wsncat()`、`wscmp()`、`wscpy()`、`wsncpy()`、`wslen()`、`wschr()`、`wsrchr()`、`wspbrk()`、`wsspn()`、`wstok()`、`windex()`、および `wrindex()` 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstod, wcstof, wcstold, wstod, watof – ワイド文字列を浮動小数点数に変換

形式

```
#include <wchar.h>

double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);

float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);

long double wcstold(const wchar_t *restrict nptr, wchar_t **restrict endptr);

double wstod(const wchar_t *nptr, wchar_t **endptr);

double watof(wchar_t *nptr);
```

機能説明

wcstod()、wcstof()、および wcstold() 関数は、*nptr* が指すワイド文字列の先頭部分を、それぞれ、double、float、および long double 型の表現に変換します。これらの関数はまず、指定されたワイド文字列を次の 3 つのシーケンスに分解します。

1. 最初のシーケンス。空白などのワイド文字から構成されます。つまり、iswspace(3C) で識別されるワイド文字です。このシーケンスは空の場合もあります。
2. 変換対象シーケンス。変換対象として認識されるワイド文字から構成されます。このシーケンスは、浮動小数点の定数として、あるいは、無限または非数を表す文字列として解釈されます。
3. 最後のシーケンス。変換対象として認識されないワイド文字から構成されます。つまり、終了ヌルなどのワイド文字です。このワイド文字は 1 文字の場合も複数文字の場合もあります。

次に、これらの関数は、変換対象シーケンスを浮動小数点数に変換して、その結果を返します。

変換対象シーケンスの期待される形式は、プラスまたはマイナス記号 (任意) の後に、次のいずれか 1 つが続くものです。

- 10 進数のシーケンス (空でない)。この後には、基数点 (任意) が続き、さらには、指数部 (任意) が続くこともあります。
- 0x または 0X とそれに続く 16 進数のシーケンス (空でない)。この後には、基数点 (任意) が続き、さらには、指数部 (任意) が続くこともあります。
- INF または INFINITY。あるいは、これと等価なワイド文字列 (大文字小文字の区別はない)。
- NAN または NAN (*n-wchar-sequence*$_{opt}$)。あるいは、これと等価なワイド文字列 (NAN 部分の大文字小文字の区別はない)。ここで、w-wchar-sequence は次のとおりです。

```
n-wchar-sequence:
    digit
    nondigit
    n-wchar-sequence digit
    n-wchar-sequence nondigit
```

wcstod() のデフォルトのモードでは、10 進数、INF/INFINITY、および、NAN/NAN (*n-char-sequence*) の形式だけが認識されます。C99/SUSv3 モードでは、これに加えて、16 進数の文字列も認識されます。

wcstod() のデフォルトのモードでは、NAN (*n-char-equence*) 形式の *n-char-sequence* 部分には、「)」(右の丸括弧) または「\」(ヌル) を除く、すべての文字を含むことができます。C99/SUSv3 モードでは、*n-char-sequence* には、英大文字、英小文字、数字、および「_」(下線) だけを含むことができます。

wcstof() 関数と wcstold() 関数は、常に、C99/SUSv3 準拠モードで動作します。

変換対象シーケンスは、「指定されたワイド文字列内で最初の空白以外のワイド文字から始まる最初で最長の部分シーケンス」であると定義されます。つまり、これを「期待される形式」と呼びます。変換対象シーケンスが期待される形式でない場合、変換対象シーケンスにワイド文字は含まれません。

変換対象シーケンスが浮動小数点数の期待される形式である場合、変換対象シーケンスは次のように解釈されます。変換対象シーケンスが数字または基数点文字から始まる場合、このシーケンスは、C 言語の規則に従って、浮動小数点の定数であると解釈されます。ただし、C 言語の規則と異なる点として、ピリオドの代わりに基数点文字が使用されます。また、変換対象シーケンスが 10 進数の浮動小数点数であり、かつ、指数部または基数点文字のどちらも存在しない場合、あるいは、変換対象シーケンスが 16 進数の浮動小数点数であり、かつ、2 進数の指数部が存在しない場合、変換対象シーケンスの最後にある数字の後に、値が 0 で適切な型の指数部が続くと想定されます。変換対象シーケンスがマイナス記号から始まる場合、このシーケンスは負であると解釈されます。変換対象シーケンスが INF または INIFITITY である場合、このシーケンスは無限であると解釈されます。変換対象シーケンスが NAN または NAN (*n-wchar-sequence*$_{opt}$) である場合、このシーケンスは非数であると解釈されます。*endptr* が指すオブジェクトには、最後のシーケンスへのポインタが格納されます。ただし、*endptr* がヌルポインタである場合を除きます。

変換対象シーケンスが 10 進数または 16 進数の形式である場合、変換の結果の値は、一般的な浮動小数点丸め方向モードに従って正しく丸められます。変換の結果、場合によっては、浮動小数点の不正確な例外、アンダーフローの例外、またはオーバーフローの例外が発生することもあります。

基数点文字は、プログラムのロケール (カテゴリ LC_NUMERIC) で定義されます。POSIX ロケールなど、基数点文字が定義されていないロケールでは、基数点文字のデフォルトはピリオド (.) です。

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。

wcstod() 関数は、正常終了した場合、errno の設定を変更しません。

wstod() 関数は、wcstod() と同等です。

watof(*str*) 関数は wstod(str, (wchar_t **)NULL) と同じです。

戻り値

これらの関数は、正常終了した場合、変換した値を返します。変換しなかった場合は、0 を返します。

変換された値が表現可能な値の範囲を超えてしまった場合、(値の符号によって)、±HUGE_VAL、±HUGE_VALF、または、±HUGE_VALL が返されます。そして、浮動小数点オーバーフローの例外が発生して、errno は ERANGE に設定されます。

値を変換した結果としてアンダーフローの例外が発生した場合、正しく丸められた結果 (通常、通常以下、または 0) が返されます。そして、浮動小数点アンダーフローの例外が発生して、errno は ERANGE 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()、wcstof()、wcstold() 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
インタフェースの安定性	wcstod()、wcstof()、および wcstold() は標準

関連項目

iswspace(3C), localeconv(3C), scanf(3C), setlocale(3C), wcstol(3C), attributes(5), standards(5)

名前

wcstod, wcstof, wcstold, wstod, watof – ワイド文字列を浮動小数点数に変換

形式

```
#include <wchar.h>

double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);

float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);

long double wcstold(const wchar_t *restrict nptr, wchar_t **restrict endptr);

double wstod(const wchar_t *nptr, wchar_t **endptr);

double watof(wchar_t *nptr);
```

機能説明

`wcstod()`、`wcstof()`、および`wcstold()`関数は、*nptr*が指すワイド文字列の先頭部分を、それぞれ、`double`、`float`、および`long double`型の表現に変換します。これらの関数はまず、指定されたワイド文字列を次の3つのシーケンスに分解します。

1. 最初のシーケンス。空白などのワイド文字から構成されます。つまり、iswspace(3C)で識別されるワイド文字です。このシーケンスは空の場合もあります。
2. 変換対象シーケンス。変換対象として認識されるワイド文字から構成されます。このシーケンスは、浮動小数点の定数として、あるいは、無限または非数を表す文字列として解釈されます。
3. 最後のシーケンス。変換対象として認識されないワイド文字から構成されます。つまり、終了ヌルなどのワイド文字です。このワイド文字は1文字の場合も複数文字の場合もあります。

次に、これらの関数は、変換対象シーケンスを浮動小数点数に変換して、その結果を返します。

変換対象シーケンスの期待される形式は、プラスまたはマイナス記号(任意)の後に、次のいずれか1つが続くものです。

- 10進数のシーケンス(空でない)。この後には、基数点(任意)が続き、さらには、指数部(任意)が続くこともあります。
- 0xまたは0Xとそれに続く16進数のシーケンス(空でない)。この後には、基数点(任意)が続き、さらには、指数部(任意)が続くこともあります。
- INFまたはINFINITY。あるいは、これと等価なワイド文字列(大文字小文字の区別はない)。
- NANまたはNAN(*n-wchar-sequence*$_{opt}$)。あるいは、これと等価なワイド文字列(NAN部分の大文字小文字の区別はない)。ここで、w-wchar-sequenceは次のとおりです。

```
n-wchar-sequence:
    digit
    nondigit
    n-wchar-sequence digit
    n-wchar-sequence nondigit
```

wcstod() のデフォルトのモードでは、10 進数、INF/INFINITY、および、NAN/NAN (*n-char-sequence*) の形式だけが認識されます。C99/SUSv3 モードでは、これに加えて、16 進数の文字列も認識されます。

wcstod() のデフォルトのモードでは、NAN (*n-char-equence*) 形式の *n-char-sequence* 部分には、「)」(右の丸括弧) または「\」(ヌル) を除く、すべての文字を含むことができます。C99/SUSv3 モードでは、*n-char-sequence* には、英大文字、英小文字、数字、および「_」(下線) だけを含むことができます。

wcstof() 関数と wcstold() 関数は、常に、C99/SUSv3 準拠モードで動作します。

変換対象シーケンスは、「指定されたワイド文字列内で最初の空白以外のワイド文字から始まる最初で最長の部分シーケンス」であると定義されます。つまり、これを「期待される形式」と呼びます。変換対象シーケンスが期待される形式でない場合、変換対象シーケンスにワイド文字は含まれません。

変換対象シーケンスが浮動小数点数の期待される形式である場合、変換対象シーケンスは次のように解釈されます。変換対象シーケンスが数字または基数点文字から始まる場合、このシーケンスは、C 言語の規則に従って、浮動小数点の定数であると解釈されます。ただし、C 言語の規則と異なる点として、ピリオドの代わりに基数点文字が使用されます。また、変換対象シーケンスが 10 進数の浮動小数点数であり、かつ、指数部または基数点文字のどちらも存在しない場合、あるいは、変換対象シーケンスが 16 進数の浮動小数点数であり、かつ、2 進数の指数部が存在しない場合、変換対象シーケンスの最後にある数字の後に、値が 0 で適切な型の指数部が続くと想定されます。変換対象シーケンスがマイナス記号から始まる場合、このシーケンスは負であると解釈されます。変換対象シーケンスが INF または INIFITITY である場合、このシーケンスは無限であると解釈されます。変換対象シーケンスが NAN または NAN (*n-wchar-sequence*$_{opt}$) である場合、このシーケンスは非数であると解釈されます。*endptr* が指すオブジェクトには、最後のシーケンスへのポインタが格納されます。ただし、*endptr* がヌルポインタである場合を除きます。

変換対象シーケンスが 10 進数または 16 進数の形式である場合、変換の結果の値は、一般的な浮動小数点丸め方向モードに従って正しく丸められます。変換の結果、場合によっては、浮動小数点の不正確な例外、アンダーフローの例外、またはオーバーフローの例外が発生することもあります。

基数点文字は、プログラムのロケール (カテゴリ LC_NUMERIC) で定義されます。POSIX ロケールなど、基数点文字が定義されていないロケールでは、基数点文字のデフォルトはピリオド (.) です。

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。

wcstod() 関数は、正常終了した場合、errno の設定を変更しません。

wstod() 関数は、wcstod() と同等です。

watof(*str*) 関数は wstod(str, (wchar_t **)NULL) と同じです。

戻り値

これらの関数は、正常終了した場合、変換した値を返します。変換しなかった場合は、0 を返します。

変換された値が表現可能な値の範囲を超えてしまった場合、(値の符号によって)、±HUGE_VAL、±HUGE_VALF、または、±HUGE_VALL が返されます。そして、浮動小数点オーバーフローの例外が発生して、errno は ERANGE に設定されます。

値を変換した結果としてアンダーフローの例外が発生した場合、正しく丸められた結果 (通常、通常以下、または 0) が返されます。そして、浮動小数点アンダーフローの例外が発生して、errno は ERANGE 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()、wcstof()、wcstold() 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
インタフェースの安定性	wcstod()、wcstof()、および wcstold() は標準

関連項目

iswspace(3C), localeconv(3C), scanf(3C), setlocale(3C), wcstol(3C), attributes(5), standards(5)

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

`malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

注意事項

`wcscat()`、`wcsncat()`、`wcscmp()`、`wcsncmp()`、`wcscpy()`、`wcsncpy()`、`wcslen()`、`wcschr()`、`wcsrchr()`、`wcspbrk()`、`wcswcs()`、`wcsspn()`、`wcscspn()`、および `wcstok()` 関数は「標準」です。`wscat()`、`wsncat()`、`wscmp()`、`wscpy()`、`wsncpy()`、`wslen()`、`wschr()`、`wsrchr()`、`wspbrk()`、`wsspn()`、`wstok()`、`windex()`、および `wrindex()` 関数は「安定」です。

名前

wcstol, wcstoll, wstol, watol, watoll, watoi – ワイド文字列を整数に変換

形式

```
#include <wchar.h>

long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr, int base);

long long wcstoll(const wchar_t *restrict nptr, wchar_t **restrict endptr, int
     base);

#include <widec.h>

long 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 `wcstoll()` の両関数は、*nptr* が示すワイド文字列の先頭部分を `long` および `long long` 型に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。

1. 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。
2. 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。
3. 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 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 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。

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。

これらの関数は、正常終了した場合、`errno` の設定を変更しません。

エラー時には、`0`、{`LONG_MIN`} または {`LLONG_MIN`}、および {`LONG_MAX`} または {`LLONG_MAX`} が返され、成功時には、有効な値が返されるため、エラー状態をチェックしたいアプリケーションは、これらの関数を呼び出す前に、`errno` を 0 に設定しておく必要があります。すると、後で `errno` をチェックできます。

`wstol()` 関数は、`wcstol()` と同等です。

`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、これらの関数は、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_MIN`}、{`LONG_MAX`}、{`LLONG_MIN`} または {`LLONG_MAX`} 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

これらの関数は、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

またこれらの関数は、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
インタフェースの安定性	wcstol() と wcstoll() は標準

関連項目

iswalpha(3C), iswspace(3C), scanf(3C), wcstod(3C), attributes(5), standards(5)

注意事項

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

wcstod, wcstof, wcstold, wstod, watof – ワイド文字列を浮動小数点数に変換

形式

```
#include <wchar.h>

double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);

float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);

long double wcstold(const wchar_t *restrict nptr, wchar_t **restrict endptr);

double wstod(const wchar_t *nptr, wchar_t **endptr);

double watof(wchar_t *nptr);
```

機能説明

wcstod()、wcstof()、およびwcstold()関数は、*nptr*が指すワイド文字列の先頭部分を、それぞれ、double、float、および long double 型の表現に変換します。これらの関数はまず、指定されたワイド文字列を次の 3 つのシーケンスに分解します。

1. 最初のシーケンス。空白などのワイド文字から構成されます。つまり、iswspace(3C) で識別されるワイド文字です。このシーケンスは空の場合もあります。
2. 変換対象シーケンス。変換対象として認識されるワイド文字から構成されます。このシーケンスは、浮動小数点の定数として、あるいは、無限または非数を表す文字列として解釈されます。
3. 最後のシーケンス。変換対象として認識されないワイド文字から構成されます。つまり、終了ヌルなどのワイド文字です。このワイド文字は 1 文字の場合も複数文字の場合もあります。

次に、これらの関数は、変換対象シーケンスを浮動小数点数に変換して、その結果を返します。

変換対象シーケンスの期待される形式は、プラスまたはマイナス記号 (任意) の後に、次のいずれか 1 つが続くものです。

- 10 進数のシーケンス (空でない)。この後には、基数点 (任意) が続き、さらには、指数部 (任意) が続くこともあります。
- 0x または 0X とそれに続く 16 進数のシーケンス (空でない)。この後には、基数点 (任意) が続き、さらには、指数部 (任意) が続くこともあります。
- INF または INFINITY。あるいは、これと等価なワイド文字列 (大文字小文字の区別はない)。
- NAN または NAN (*n-wchar-sequence*$_{opt}$)。あるいは、これと等価なワイド文字列 (NAN 部分の大文字小文字の区別はない)。ここで、w-wchar-sequence は次のとおりです。

```
n-wchar-sequence:
    digit
    nondigit
    n-wchar-sequence digit
    n-wchar-sequence nondigit
```

wcstod() のデフォルトのモードでは、10 進数、INF/INFINITY、および、NAN/NAN (*n-char-sequence*) の形式だけが認識されます。C99/SUSv3 モードでは、これに加えて、16 進数の文字列も認識されます。

wcstod() のデフォルトのモードでは、NAN (*n-char-equence*) 形式の *n-char-sequence* 部分には、「)」(右の丸括弧) または「\」(ヌル) を除く、すべての文字を含むことができます。C99/SUSv3 モードでは、*n-char-sequence* には、英大文字、英小文字、数字、および「_」(下線) だけを含むことができます。

wcstof() 関数と wcstold() 関数は、常に、C99/SUSv3 準拠モードで動作します。

変換対象シーケンスは、「指定されたワイド文字列内で最初の空白以外のワイド文字から始まる最初で最長の部分シーケンス」であると定義されます。つまり、これを「期待される形式」と呼びます。変換対象シーケンスが期待される形式でない場合、変換対象シーケンスにワイド文字は含まれません。

変換対象シーケンスが浮動小数点数の期待される形式である場合、変換対象シーケンスは次のように解釈されます。変換対象シーケンスが数字または基数点文字から始まる場合、このシーケンスは、C 言語の規則に従って、浮動小数点の定数であると解釈されます。ただし、C 言語の規則と異なる点として、ピリオドの代わりに基数点文字が使用されます。また、変換対象シーケンスが 10 進数の浮動小数点数であり、かつ、指数部または基数点文字のどちらも存在しない場合、あるいは、変換対象シーケンスが 16 進数の浮動小数点数であり、かつ、2 進数の指数部が存在しない場合、変換対象シーケンスの最後にある数字の後に、値が 0 で適切な型の指数部が続くと想定されます。変換対象シーケンスがマイナス記号から始まる場合、このシーケンスは負であると解釈されます。変換対象シーケンスが INF または INIFITITY である場合、このシーケンスは無限であると解釈されます。変換対象シーケンスが NAN または NAN (*n-wchar-sequence*$_{opt}$) である場合、このシーケンスは非数であると解釈されます。*endptr* が指すオブジェクトには、最後のシーケンスへのポインタが格納されます。ただし、*endptr* がヌルポインタである場合を除きます。

変換対象シーケンスが 10 進数または 16 進数の形式である場合、変換の結果の値は、一般的な浮動小数点丸め方向モードに従って正しく丸められます。変換の結果、場合によっては、浮動小数点の不正確な例外、アンダーフローの例外、またはオーバーフローの例外が発生することもあります。

基数点文字は、プログラムのロケール (カテゴリ LC_NUMERIC) で定義されます。POSIX ロケールなど、基数点文字が定義されていないロケールでは、基数点文字のデフォルトはピリオド (.) です。

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。

wcstod() 関数は、正常終了した場合、errno の設定を変更しません。

wstod() 関数は、wcstod() と同等です。

watof(*str*) 関数は wstod(str, (wchar_t **)NULL) と同じです。

戻り値

これらの関数は、正常終了した場合、変換した値を返します。変換しなかった場合は、0 を返します。

変換された値が表現可能な値の範囲を超えてしまった場合、(値の符号によって)、±HUGE_VAL、±HUGE_VALF、または、±HUGE_VALL が返されます。そして、浮動小数点オーバーフローの例外が発生して、errno は ERANGE に設定されます。

値を変換した結果としてアンダーフローの例外が発生した場合、正しく丸められた結果 (通常、通常以下、または 0) が返されます。そして、浮動小数点アンダーフローの例外が発生して、errno は ERANGE 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()、wcstof()、wcstold() 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
インタフェースの安定性	wcstod()、wcstof()、および wcstold() は標準

関連項目

iswspace(3C), localeconv(3C), scanf(3C), setlocale(3C), wcstol(3C), attributes(5), standards(5)

名前

wcstol, wcstoll, wstol, watol, watoll, watoi – ワイド文字列を整数に変換

形式

```
#include <wchar.h>

long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr, int base);

long long wcstoll(const wchar_t *restrict nptr, wchar_t **restrict endptr, int
     base);

#include <widec.h>

long 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 `wcstoll()` の両関数は、*nptr* が示すワイド文字列の先頭部分を `long` および `long long` 型に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。

1. 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。
2. 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。
3. 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 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 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。

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。

これらの関数は、正常終了した場合、`errno` の設定を変更しません。

エラー時には、`0`、{`LONG_MIN`} または {`LLONG_MIN`}、および {`LONG_MAX`} または {`LLONG_MAX`} が返され、成功時には、有効な値が返されるため、エラー状態をチェックしたいアプリケーションは、これらの関数を呼び出す前に、`errno` を 0 に設定しておく必要があります。すると、後で `errno` をチェックできます。

`wstol()` 関数は、`wcstol()` と同等です。

`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、これらの関数は、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_MIN`}、{`LONG_MAX`}、{`LLONG_MIN`} または {`LLONG_MAX`} 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

これらの関数は、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

またこれらの関数は、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
インタフェースの安定性	wcstol() と wcstoll() は標準

関連項目

iswalpha(3C), iswspace(3C), scanf(3C), wcstod(3C), attributes(5), standards(5)

注意事項

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

wcstombs – ワイド文字列を文字列に変換

形式

```
#include <stdlib.h>

size_t wcstombs(char *restrict s, const wchar_t *restrict 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 バイト以下の場合は、*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。それ以外の場合は、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	Enabled
インタフェースの安定性	標準

関連項目

mblen(3C), mbstowcs(3C), mbtowc(3C), setlocale(3C), wctomb(3C), attributes(5), standards(5)

名前

wcstoul, wcstoull – 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 wcstoul(const wchar_t *restrict nptr, wchar_t **restrict endptr,
     int base);

unsigned long long wcstoull(const wchar_t *restrict nptr, wchar_t **restrict
     endptr, int base);
```

機能説明

wcstoul() と wcstoull() 関数は、*nptr* が示すワイド文字列の先頭部分を unsigned long と unsigned long long 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。

1. 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。
2. 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。
3. 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して、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 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 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。

変換対象シーケンスが空であるか、期待される形式でない場合、変換は実行されません。*endptr* が指すオブジェクトには、*nptr* の値が格納されます。ただし、*endptr* がヌルポインタである場合を除きます。

`wcstoul()` 関数は、正常終了した場合、`errno` の設定を変更しません。

エラー時には、0、{`ULONG_MAX`}、および {`ULLONG_MAX`} が返され、成功時には、有効な値が返されるため、エラー状態をチェックしたいアプリケーションは、`wcstoul()` または `wcstoull()` を呼び出す前に、`errno` を 0 に設定しておく必要があります。これにより、後で `errno` をチェックできます。

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()` と`wcstoull()` 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()` と`wcstoull()` 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()` と`wcstoull()` 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()` または `wcstoull()` の範囲外数値として使用します。`wcstoul()` または `wcstoull()` 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
インタフェースの安定性	標準

関連項目

isspace(3C), iswalpha(3C), scanf(3C), wcstod(3C), wcstol(3C), attributes(5), standards(5)

名前

wcstoul, wcstoull – 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 wcstoul(const wchar_t *restrict nptr, wchar_t **restrict endptr,
     int base);

unsigned long long wcstoull(const wchar_t *restrict nptr, wchar_t **restrict
     endptr, int base);
```

機能説明

`wcstoul()` と `wcstoull()` 関数は、*nptr* が示すワイド文字列の先頭部分を `unsigned long` と `unsigned long long` 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。

1. 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。
2. 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。
3. 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して、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 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 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。

変換対象シーケンスが空であるか、期待される形式でない場合、変換は実行されません。*endptr* が指すオブジェクトには、*nptr* の値が格納されます。ただし、*endptr* がヌルポインタである場合を除きます。

wcstoul() 関数は、正常終了した場合、errno の設定を変更しません。

エラー時には、0、{ULONG_MAX}、および {ULLONG_MAX} が返され、成功時には、有効な値が返されるため、エラー状態をチェックしたいアプリケーションは、wcstoul() または wcstoull() を呼び出す前に、errno を 0 に設定しておく必要があります。これにより、後で errno をチェックできます。

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() と wcstoull() 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() と wcstoull() 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() と wcstoull() 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() または wcstoull() の範囲外数値として使用します。wcstoul() または wcstoull() 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
インタフェースの安定性	標準

関連項目

isspace(3C), iswalpha(3C), scanf(3C), wcstod(3C), wcstol(3C), attributes(5), standards(5)

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

デフォルトとその他の標準仕様

```
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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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かれば 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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つかれば 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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

`malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

注意事項

`wcscat()`、`wcsncat()`、`wcscmp()`、`wcsncmp()`、`wcscpy()`、`wcsncpy()`、`wcslen()`、`wcschr()`、`wcsrchr()`、`wcspbrk()`、`wcswcs()`、`wcsspn()`、`wcscspn()`、および `wcstok()` 関数は「標準」です。`wscat()`、`wsncat()`、`wscmp()`、`wscpy()`、`wsncpy()`、`wslen()`、`wschr()`、`wsrchr()`、`wspbrk()`、`wsspn()`、`wstok()`、`windex()`、および `wrindex()` 関数は「安定」です。

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、*pws* 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
インタフェースの安定性	標準

関連項目

setlocale(3C), wcwidth(3C), attributes(5), standards(5)

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。

wcsxfrm() 関数と wsxfrm() 関数は、正常終了した場合、errno の設定を変更しません。

エラーを示す戻り値は予約されていないため、エラー状態をチェックしたいアプリケーションは、wcsxfrm() または wsxfrm() を呼び出す前に、errno を 0 に設定しておく必要があります。すると、後で errno をチェックできます。

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。

エラー時、wcsxfrm() と wsxfrm() は errno を設定できますが、エラーを示す戻り値は予約されていません。

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。

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
インタフェースの安定性	`wcsxfrm()` は標準

関連項目

`setlocale`(3C), `wcscmp`(3C), `wcscoll`(3C), `wscmp`(3C), `wscoll`(3C), `attributes`(5), `standards`(5)

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なっていない限り、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
インタフェースの安定性	標準

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), standards(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
インタフェースの安定性	標準

関連項目

iswctype(3C), setlocale(3C), attributes(5), standards(5)

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
インタフェースの安定性	標準

関連項目　setlocale(3C), wcswidth(3C), attributes(5), standards(5)

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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かれば 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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つかれば 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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

`malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

注意事項

`wcscat()`、`wcsncat()`、`wcscmp()`、`wcsncmp()`、`wcscpy()`、`wcsncpy()`、`wcslen()`、`wcschr()`、`wcsrchr()`、`wcspbrk()`、`wcswcs()`、`wcsspn()`、`wcscspn()`、および `wcstok()` 関数は「標準」です。`wscat()`、`wsncat()`、`wscmp()`、`wscpy()`、`wsncpy()`、`wslen()`、`wschr()`、`wsrchr()`、`wspbrk()`、`wsspn()`、`wstok()`、`windex()`、および `wrindex()` 関数は「安定」です。

名前

wstring, wscasecmp, wsncasecmp, 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 wsncasecmp(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( ), wsncasecmp( )

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。wsncasecmp() も同じ比較を行いますが、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)

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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かれば 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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つかれば 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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

`malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

注意事項

`wcscat()`、`wcsncat()`、`wcscmp()`、`wcsncmp()`、`wcscpy()`、`wcsncpy()`、`wcslen()`、`wcschr()`、`wcsrchr()`、`wcspbrk()`、`wcswcs()`、`wcsspn()`、`wcscspn()`、および `wcstok()` 関数は「標準」です。`wscat()`、`wsncat()`、`wscmp()`、`wscpy()`、`wsncpy()`、`wslen()`、`wschr()`、`wsrchr()`、`wspbrk()`、`wsspn()`、`wstok()`、`windex()`、および `wrindex()` 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

`malloc`(3C), `string`(3C), `wcswidth`(3C), `wcwidth`(3C), `attributes`(5), `standards`(5)

注意事項

`wcscat()`、`wcsncat()`、`wcscmp()`、`wcsncmp()`、`wcscpy()`、`wcsncpy()`、`wcslen()`、`wcschr()`、`wcsrchr()`、`wcspbrk()`、`wcswcs()`、`wcsspn()`、`wcscspn()`、および `wcstok()` 関数は「標準」です。`wscat()`、`wsncat()`、`wscmp()`、`wscpy()`、`wsncpy()`、`wslen()`、`wschr()`、`wsrchr()`、`wspbrk()`、`wsspn()`、`wstok()`、`windex()`、および `wrindex()` 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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かれば 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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つかれば 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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wstring, wscasecmp, wsncasecmp, 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 wsncasecmp(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( ), wsncasecmp( )

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。wsncasecmp() も同じ比較を行いますが、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)

名前

wcscoll, 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。

`wcscoll()` 関数と `wscoll()` 関数は、正常終了した場合、`errno` の設定を変更しません。

エラー状態をチェックしたいアプリケーションは、`wcscoll()` または `wscoll()` を呼び出す前に、`errno` を 0 に設定しておく必要があります。0 以外の `errno` が返った場合、エラーが発生していたことになります。

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

使用法

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
インタフェースの安定性	`wcscoll()` は標準

関連項目

`setlocale(3C)`, `wcscmp(3C)`, `wcsxfrm(3C)`, `attributes(5)`, `standards(5)`

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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かれば 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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つかれば 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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

`malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

注意事項

`wcscat()`、`wcsncat()`、`wcscmp()`、`wcsncmp()`、`wcscpy()`、`wcsncpy()`、`wcslen()`、`wcschr()`、`wcsrchr()`、`wcspbrk()`、`wcswcs()`、`wcsspn()`、`wcscspn()`、および `wcstok()` 関数は「標準」です。`wscat()`、`wsncat()`、`wscmp()`、`wscpy()`、`wsncpy()`、`wslen()`、`wschr()`、`wsrchr()`、`wspbrk()`、`wsspn()`、`wstok()`、`windex()`、および `wrindex()` 関数は「安定」です。

名前

wstring, wscasecmp, wsncasecmp, 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 wsncasecmp(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(), wsncasecmp()

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。wsncasecmp() も同じ比較を行いますが、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)

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wstring, wscasecmp, wsncasecmp, 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 wsncasecmp(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( ), wsncasecmp( )

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。wsncasecmp() も同じ比較を行いますが、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)

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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かれば 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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つかれば 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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

`malloc(3C)`, `string(3C)`, `wcswidth(3C)`, `wcwidth(3C)`, `attributes(5)`, `standards(5)`

注意事項

`wcscat()`、`wcsncat()`、`wcscmp()`、`wcsncmp()`、`wcscpy()`、`wcsncpy()`、`wcslen()`、`wcschr()`、`wcsrchr()`、`wcspbrk()`、`wcswcs()`、`wcsspn()`、`wcscspn()`、および `wcstok()` 関数は「標準」です。`wscat()`、`wsncat()`、`wscmp()`、`wscpy()`、`wsncpy()`、`wslen()`、`wschr()`、`wsrchr()`、`wspbrk()`、`wsspn()`、`wstok()`、`windex()`、および `wrindex()` 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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かれば 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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つかれば 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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wsprintf – 書式付き出力変換

形式

```
#include <stdio.h>
#include <widec.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, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wsscanf – 書式付き入力変換

形式

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

wsprintf(3C), printf(3C), scanf(3C), attributes(5)

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstod, wcstof, wcstold, wstod, watof – ワイド文字列を浮動小数点数に変換

形式

```
#include <wchar.h>

double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);

float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);

long double wcstold(const wchar_t *restrict nptr, wchar_t **restrict endptr);

double wstod(const wchar_t *nptr, wchar_t **endptr);

double watof(wchar_t *nptr);
```

機能説明

`wcstod()`、`wcstof()`、および`wcstold()`関数は、*nptr*が指すワイド文字列の先頭部分を、それぞれ、`double`、`float`、および`long double`型の表現に変換します。これらの関数はまず、指定されたワイド文字列を次の3つのシーケンスに分解します。

1. 最初のシーケンス。空白などのワイド文字から構成されます。つまり、iswspace(3C)で識別されるワイド文字です。このシーケンスは空の場合もあります。
2. 変換対象シーケンス。変換対象として認識されるワイド文字から構成されます。このシーケンスは、浮動小数点の定数として、あるいは、無限または非数を表す文字列として解釈されます。
3. 最後のシーケンス。変換対象として認識されないワイド文字から構成されます。つまり、終了ヌルなどのワイド文字です。このワイド文字は1文字の場合も複数文字の場合もあります。

次に、これらの関数は、変換対象シーケンスを浮動小数点数に変換して、その結果を返します。

変換対象シーケンスの期待される形式は、プラスまたはマイナス記号(任意)の後に、次のいずれか1つが続くものです。

- 10進数のシーケンス(空でない)。この後には、基数点(任意)が続き、さらには、指数部(任意)が続くこともあります。
- 0xまたは0Xとそれに続く16進数のシーケンス(空でない)。この後には、基数点(任意)が続き、さらには、指数部(任意)が続くこともあります。
- INFまたはINFINITY。あるいは、これと等価なワイド文字列(大文字小文字の区別はない)。
- NANまたはNAN(*n-wchar-sequence*$_{opt}$)。あるいは、これと等価なワイド文字列(NAN部分の大文字小文字の区別はない)。ここで、w-wchar-sequenceは次のとおりです。

```
n-wchar-sequence:
    digit
    nondigit
    n-wchar-sequence digit
    n-wchar-sequence nondigit
```

wcstod() のデフォルトのモードでは、10 進数、INF/INFINITY、および、NAN/NAN (*n-char-sequence*) の形式だけが認識されます。C99/SUSv3 モードでは、これに加えて、16 進数の文字列も認識されます。

wcstod() のデフォルトのモードでは、NAN (*n-char-equence*) 形式の *n-char-sequence* 部分には、「)」(右の丸括弧) または「\」(ヌル) を除く、すべての文字を含むことができます。C99/SUSv3 モードでは、*n-char-sequence* には、英大文字、英小文字、数字、および「_」(下線) だけを含むことができます。

wcstof() 関数と wcstold() 関数は、常に、C99/SUSv3 準拠モードで動作します。

変換対象シーケンスは、「指定されたワイド文字列内で最初の空白以外のワイド文字から始まる最初で最長の部分シーケンス」であると定義されます。つまり、これを「期待される形式」と呼びます。変換対象シーケンスが期待される形式でない場合、変換対象シーケンスにワイド文字は含まれません。

変換対象シーケンスが浮動小数点数の期待される形式である場合、変換対象シーケンスは次のように解釈されます。変換対象シーケンスが数字または基数点文字から始まる場合、このシーケンスは、C 言語の規則に従って、浮動小数点の定数であると解釈されます。ただし、C 言語の規則と異なる点として、ピリオドの代わりに基数点文字が使用されます。また、変換対象シーケンスが 10 進数の浮動小数点数であり、かつ、指数部または基数点文字のどちらも存在しない場合、あるいは、変換対象シーケンスが 16 進数の浮動小数点数であり、かつ、2 進数の指数部が存在しない場合、変換対象シーケンスの最後にある数字の後に、値が 0 で適切な型の指数部が続くと想定されます。変換対象シーケンスがマイナス記号から始まる場合、このシーケンスは負であると解釈されます。変換対象シーケンスが INF または INIFITITY である場合、このシーケンスは無限であると解釈されます。変換対象シーケンスが NAN または NAN (*n-wchar-sequence*$_{opt}$) である場合、このシーケンスは非数であると解釈されます。*endptr* が指すオブジェクトには、最後のシーケンスへのポインタが格納されます。ただし、*endptr* がヌルポインタである場合を除きます。

変換対象シーケンスが 10 進数または 16 進数の形式である場合、変換の結果の値は、一般的な浮動小数点丸め方向モードに従って正しく丸められます。変換の結果、場合によっては、浮動小数点の不正確な例外、アンダーフローの例外、またはオーバーフローの例外が発生することもあります。

基数点文字は、プログラムのロケール (カテゴリ LC_NUMERIC) で定義されます。POSIX ロケールなど、基数点文字が定義されていないロケールでは、基数点文字のデフォルトはピリオド (.) です。

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。

wcstod() 関数は、正常終了した場合、errno の設定を変更しません。

wstod() 関数は、wcstod() と同等です。

watof(*str*) 関数は wstod(str, (wchar_t **)NULL) と同じです。

戻り値

これらの関数は、正常終了した場合、変換した値を返します。変換しなかった場合は、0 を返します。

変換された値が表現可能な値の範囲を超えてしまった場合、(値の符号によって)、±HUGE_VAL、±HUGE_VALF、または、±HUGE_VALL が返されます。そして、浮動小数点オーバーフローの例外が発生して、errno は ERANGE に設定されます。

値を変換した結果としてアンダーフローの例外が発生した場合、正しく丸められた結果 (通常、通常以下、または 0) が返されます。そして、浮動小数点アンダーフローの例外が発生して、errno は ERANGE 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()、wcstof()、wcstold() 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
インタフェースの安定性	wcstod()、wcstof()、および wcstold() は標準

関連項目

iswspace(3C), localeconv(3C), scanf(3C), setlocale(3C), wcstol(3C), attributes(5), standards(5)

名前

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – ワイド文字列の操作

形式

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcscpy(wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict 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 *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

wchar_t *wcswcs(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);
```

XPG4, SUS, SUSv2, SUSv3 デフォルトとその他の標準仕様

```
wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict 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 wsspn(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 *windex(const wchar_t *ws, wchar_t wc);

wchar_t *wrindex(const wchar_t *ws, wchar_t wc);
```

ISO C++

```
#include <wchar.h>

const wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

const wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

const wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

#include <cwchar>

wchar_t *std::wcschr(wchar_t *ws, wchar_t wc);

wchar_t *std::wcspbrk(wchar_t *ws1, const wchar_t *ws2);

wchar_t *std::wcsrchr(wchar_t *ws, wchar_t wc);
```

機能説明

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。

wcscat( ), wscat( )

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。

wcsncat( ), wsncat( )

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。

wcscmp( ), wscmp( )

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。

wcsncmp( ), wsncmp( ) `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。

wcscpy( ), wscpy( ) `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。

wcsncpy( ), wsncpy( ) `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。

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( ), wschr( ) `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か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か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 `wschr()` と `wsrchr()` 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か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。

wcswcs( )

`wcswcs()` 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 (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、*ws2* 中のワイド文字列が見つか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。*ws2* が長さ 0 のワイド文字列を指している場合、関数は *ws1* を返します。

wcsspn( ), wsspn( )

`wcsspn()` と `wsspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcscspn( ), wscspn( )

`wcscspn()` と `wscspn()` の両関数は、*ws1* が指すワイド文字列を先頭から検査し、*ws2* 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 *ws1* 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。

wcstok( ), wstok( )

`wcstok()` と `wstok()` を連続して呼び出すと、*ws1* 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、*ws2* 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

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)。

最初の呼び出しでは *ws1* 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 *ws2* 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。

1 回目の呼び出しでは、*ws1* が示すワイド文字列を先頭から検査して、*ws2* 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、*ws1* 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。

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。*ws1* 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かれば、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。

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
インタフェースの安定性	「注意事項」を参照。

関連項目

malloc(3C), string(3C), wcswidth(3C), wcwidth(3C), attributes(5), standards(5)

注意事項

wcscat()、wcsncat()、wcscmp()、wcsncmp()、wcscpy()、wcsncpy()、wcslen()、wcschr()、wcsrchr()、wcspbrk()、wcswcs()、wcsspn()、wcscspn()、および wcstok() 関数は「標準」です。wscat()、wsncat()、wscmp()、wscpy()、wsncpy()、wslen()、wschr()、wsrchr()、wspbrk()、wsspn()、wstok()、windex()、および wrindex() 関数は「安定」です。

名前

wcstol, wcstoll, wstol, watol, watoll, watoi – ワイド文字列を整数に変換

形式

```
#include <wchar.h>

long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr, int base);

long long wcstoll(const wchar_t *restrict nptr, wchar_t **restrict endptr, int
     base);

#include <widec.h>

long 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 `wcstoll()` の両関数は、*nptr* が示すワイド文字列の先頭部分を `long` および `long long` 型に変換します。まず、指定されたワイド文字列を 3 つの部分に分けます。

1. 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。
2. 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。
3. 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 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 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。

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。

これらの関数は、正常終了した場合、`errno` の設定を変更しません。

エラー時には、0、{`LONG_MIN`} または {`LLONG_MIN`}、および {`LONG_MAX`} または {`LLONG_MAX`} が返され、成功時には、有効な値が返されるため、エラー状態をチェックしたいアプリケーションは、これらの関数を呼び出す前に、`errno` を 0 に設定しておく必要があります。すると、後で `errno` をチェックできます。

`wstol()` 関数は、`wcstol()` と同等です。

`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、これらの関数は、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_MIN`}、{`LONG_MAX`}、{`LLONG_MIN`} または {`LLONG_MAX`} 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

これらの関数は、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

またこれらの関数は、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
インタフェースの安定性	wcstol() と wcstoll() は標準

関連項目

iswalpha(3C), iswspace(3C), scanf(3C), wcstod(3C), attributes(5), standards(5)

注意事項

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, wsncasecmp, 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 wsncasecmp(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( ), wsncasecmp( )

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。wsncasecmp() も同じ比較を行いますが、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)

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。

wcsxfrm() 関数と wsxfrm() 関数は、正常終了した場合、errno の設定を変更しません。

エラーを示す戻り値は予約されていないため、エラー状態をチェックしたいアプリケーションは、wcsxfrm() または wsxfrm() を呼び出す前に、errno を 0 に設定しておく必要があります。すると、後で errno をチェックできます。

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。

エラー時、wcsxfrm() と wsxfrm() は errno を設定できますが、エラーを示す戻り値は予約されていません。

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。

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
インタフェースの安定性	`wcsxfrm()` は標準

関連項目

`setlocale`(3C), `wcscmp`(3C), `wcscoll`(3C), `wscmp`(3C), `wscoll`(3C), `attributes`(5), `standards`(5)