LAPIS SEMICONDUCTOR

SQ003077E001

RTLU8 ランタイムライブラリ リファレンス

2011 年 10 月発行

ご注意

本資料の一部または全部をラピスセミコンダクタの許可なく、転載・複写することを堅くお断りします。

本資料の記載内容は改良などのため予告なく変更することがあります。

本資料に記載されている内容は製品のご紹介資料です。ご使用にあたりましては、別途仕様書を必ずご請求のうえ、ご確認ください。

本資料に記載されております応用回路例やその定数などの情報につきましては、本製品の標準的な動作や使い方を説明するものです。したがいまして、量産設計をされる場合には、外部諸条件を考慮していただきますようお願いいたします。

本資料に記載されております情報は、正確を期すため慎重に作成したものですが、万が一、当該情報の誤り・誤植に起因する損害がお客様に生じた場合においても、ラピスセミコンダクタはその責任を負うものではありません。

本資料に記載されております技術情報は、製品の代表的動作および応用回路例などを示したものであり、ラピスセミコンダクタまたは他社の知的財産権その他のあらゆる権利について明示的にも黙示的にも、その実施または利用を許諾するものではありません。上記技術情報の使用に起因して紛争が発生した場合、ラピスセミコンダクタはその責任を負うものではありません。

本資料に掲載されております製品は、一般的な電子機器（AV 機器、OA 機器、通信機器、家電製品、アミューズメント機器など）への使用を意図しています。

本資料に掲載されております製品は、「耐放射線設計」はなされておりません。

ラピスセミコンダクタは常に品質・信頼性の向上に取り組んでおりますが、種々の要因で故障することもあり得ます。

ラピスセミコンダクタ製品が故障した際、その影響により人身事故、火災損害等が起こらないようご使用機器でのディレーティング、冗長設計、延焼防止、フェイルセーフ等の安全確保をお願いします。定格を超えたご使用や使用上の注意書が守られていない場合、いかなる責任もラピスセミコンダクタは負うものではありません。

極めて高度な信頼性が要求され、その製品の故障や誤動作が直接人命を脅かしあるいは人体に危害を及ぼすおそれのある機器・装置・システム（医療機器、輸送機器、航空宇宙機、原子力制御、燃料制御、各種安全装置など）へのご使用を意図して設計・製造されたものではありません。上記特定用途に使用された場合、いかなる責任もラピスセミコンダクタは負うものではありません。上記特定用途への使用を検討される際は、事前にローム営業窓口までご相談願います。

本資料に記載されております製品および技術のうち「外国為替及び外国貿易法」に該当する製品または技術を輸出する場合、または国外に提供する場合には、同法に基づく許可が必要です。

Windows は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。また、その他の製品名や社名などは、一般に商標または登録商標です。

ラピスセミコンダクタ株式会社

〒193-8550 東京都八王子市東浅川町 550 番地 1

http://www.lapis-semi.com/jp/

目次

はじめに

このマニュアルの構成

このマニュアルは，RTLU8 ランタイムライブラリについて説明しています。このマニュアルは，ユーザが C 言語によるプログラミングに習熟し，nX-U8 CPU コアのアーキテクチャについて十分な知識を有していることを前提として記述されています。

本書は，次の 6 つの章で構成されています。

第 1 章　概要
RTLU8 ランタイムライブラリの概要を説明します。ここでは，RTLU8 のファイル構成，実際の使い方，マクロと関数の使い分け，ライブラリルーチンの簡単な機能について解説しています。

第 2 章　データモデル別ライブラリ関数
データモデル別に存在する RTLU8 オリジナルのライブラリルーチンについて説明します。

第 3 章　標準組み込みルーチンリファレンス
ライブラリルーチンのうち，標準の組み込みルーチンの詳細をアルファベット順に説明します。

第 4 章　標準入出力ルーチンリファレンス
ライブラリルーチンのうち，標準入出力を扱うルーチンの詳細をアルファベット順に説明します。

第 5 章　低水準関数
低水準関数 read と write について説明します。

付録
標準 C ライブラリルーチンとデータモデル別に用意したライブラリルーチンの形式の一覧です。

関連するドキュメント

必要に応じて次のドキュメントを参照してください。

CCU8 ユーザーズマニュアル

C コンパイラ CCU8 の使い方と言語仕様を解説しています。

MACU8 アセンブラパッケージユーザーズマニュアル

MACU8 アセンブラパッケージに含まれるソフトウェアの使い方とアセンブリ言語仕様を解説しています。

表記法と用語

このマニュアルでは，説明をわかりやすくするために，いくつかの記号を使用しています。このマニュアルで使用する記号とその意味は次のとおりです。

記号	意味
SAMPLE	この文字は，画面に表示されるメッセージや，コマンドラインの入力例，作成されるリストファイルの例などを示します。
itarics	斜体で表示された項目は，そのまま文字を入力するのではなく，必要な情報に置き換えて入力することを示します。
[]	[]の中身は必要に応じて入力する項目です。省略することも可能です。
…	…の直前の項目を必要に応じて繰り返すことができます。
{*choice1*|*choice2*}	中カッコ（{}）の中の縦棒（|）で区切られた項目のうち，どれか１つを選んで入力することを示します。[]で囲まれていない限り，必ず１つは入力しなければなりません。
value1～*value2*	*value1* 以上，*value2* 以下の値を示します。
PROGRAM ・ ・ ・ PROGRAM	縦に並んだ点は，プログラムの例が一部省略されていることを示します。

このマニュアル全体で使用する用語とその意味は次のとおりです。

用語	意味
マクロ	#define 前処理指令で定義される名前です。本書では，パラメータ付きのマクロのことを，簡単に“マクロ”と表現する場合があります。
ルーチン	関数とパラメータ付きのマクロを総称して，“ルーチン”と表現します。
ライブラリルーチン	RTLU8 に含まれる“ルーチン”のことです。
型	typedef によって定義される名前です。
定数マクロ	パラメータをもたず、常に一定の値を与えるマクロです。単に“定数”と表現することもあります。
null 文字	アスキーコード 0x00 の文字，すなわち‘¥0’のことです。
null 文字列	長さが 0，すなわち先頭に null 文字を持つ文字列です。
null ターミネータ	文字列の終わりとしての null 文字のことです。
null ポインタ	アドレス 0 へのポインタ。定数マクロ NULL で表現されます。

1 概要

1.1 RTLU8 ランタイムライブラリについて

RTLU8 は，nX-U8 を CPU コアとするマイクロコントローラのための C ランタイムライブラリです。RTLU8 は，C プログラミングにおいて使用される多くのルーチンを提供します。これらのライブラリルーチンを使用することによって，多くの労力と時間を節約することができます。

1.2 RTLU8 ランタイムライブラリの構成

ここでは，RTLU8 ランタイムライブラリを構成するファイルについて解説します。

RTLU8 ランタイムライブラリは，11 個のヘッダファイルと，いくつかのライブラリファイルで構成されています。

1.2.1 ヘッダファイル

ヘッダファイルは，機能別に 11 個用意されています。各ヘッダファイルには，関数のプロトタイプ宣言，マクロの定義，型の定義が含まれています。

ヘッダファイルは，ソースプログラムをコンパイルするときに必要になります。CCU8 は，ソースプログラムの中で#include 前処理指令によって指定されているヘッダファイルをインクルードします。

各ヘッダファイル名とその内容は，次のとおりです。

ヘッダファイル名	内容
ctype.h	文字の分類と変換
error.h	エラーの識別
float.h	浮動小数点の各境界値
limits.h	整数の各境界値
math.h	数値計算関数
setjmp.h	グローバルジャンプ
stdarg.h	可変引数リスト
stddef.h	標準の型とマクロ
stdio.h	入出力関連処理
stdlib.h	汎用ユーティリティ
string.h	文字列操作

1.2.2 ライブラリファイル

ライブラリファイルには，各ライブラリルーチンが含まれています。ライブラリファイルの形式は，RASU8 や RLU8 が出力するオブジェクトファイルと同様，バイナリ形式です。

ライブラリファイルは，リンク時に必要になります。RLU8 は，プログラムで使用しているライブラリルーチンをライブラリファイルの中から探して，プログラムとリンクし，アブソリュートオブジェクトファイル（.ABS）を作成します。

nX-U8 用に用意されたライブラリファイルは次のとおりです。

ライブラリファイル名	内容
LU8100SW.LIB	Small メモリモデル用ライブラリ
LU8100LW.LIB	Large メモリモデル用ライブラリ

リンク時には，CCU8 でコンパイルしたときのメモリモデルに対応するライブラリファイルを指定してください。

1.3 ANSI/ISO9899 C規格との互換性

RTLU8 は，基本的には ANSI/ISO9899 C が提唱するライブラリ仕様のサブセット版です。

次の標準ヘッダファイルは，RTLU8 には含まれていません。

ヘッダファイル名	意味
assert.h	実行時の条件チェック
locale.h	地域化の設定と変更
signal.h	シグナル処理関数
time.h	日付・時間処理関数

関数，マクロ，定数マクロ，型の名称とそのインターフェースや機能は，すべて ANSI/ISO9899 C に準拠しています。

RTLU8 にはまた，ANSI/ISO9899 C では規定されていない，いくつかの関数も含まれています。これらのオリジナル関数は，CCU8 のアーキテクチャの特徴であるデータモデル（NEAR モデル / FAR モデル）に対処するために用意されています。詳しくは「第 2 章　データモデル別のライブラリ関数」を参照してください。

1.4 ライブラリルーチンの使用方法

ここでは，RTLU8 を使用するための環境設定と，ライブラリルーチンを使用したプログラムの記述，コンパイルからリンクまでの手順を説明します。

1.4.1 環境変数INCLU8 の設定

ヘッダファイルが格納されているパス名を CCU8 に知らせるために，環境変数 INCLU8 を設定します。CCU8 は，ソースファイルで#include 前処理指令で指定されているヘッダファイルを，環境変数 INCLU8 にセットされているパスからサーチします。

環境変数 INCLU8 のセットは，DOS の SET コマンドを使用して行ないます。SET コマンドの書式は，次のとおりです。

```
SET INCLU8=path
```

例

ヘッダファイルが，C:¥U8¥INCLUDE に格納されている場合，次のように実行します。

```
SET INCLU8=C:¥U8¥INCLUDE
```

参考

ヘッダファイルのパスは，CCU8 の/I*path* オプションを使用して指定することもできます。例えば上記の例のパス指定を，/I オプションを使用して行なうと，次のようになります。

```
CCU8 /TM610001 /IC:¥U8¥INCLUDE FOO.C
```

1.4.2 プログラムの記述

各ライブラリルーチンを使用する際には，それに対応したヘッダファイルをインクルードする必要があります。必要なヘッダファイルをインクルードするためには，#include 前処理指令を使用します。CCU8 は，#include 前処理指令で指定されているヘッダファイルを，ソースファイルに挿入します。各ライブラリルーチンがどのヘッダファイルを必要とするのかについては，第 3 章以降のライブラリリファレンスを参照してください。

例 1

memcpy 関数を使用する例を示します。memcpy 関数に対応するヘッダファイルは，string.h です。したがって，ソースファイルには，次のように記述します。

```
#include <string.h>
```

例 2

複数のヘッダファイルをインクルードする場合，インクルードの順番に制限はありません。

string.h と math.h をインクルードする場合，

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

と記述しても，

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

と記述しても，どちらでもかまいません。

#include 前処理指令のファイル名の指定には，上記のように山形カッコ（<>）でファイル名を囲む方法と，二重引用符（""）でファイル名を囲む方法があります。RTLU8 のヘッダファイルをインクルードする場合は，山形カッコを使用してください。#include 前処理指令の詳細については，『CCU8 ユーザーズマニュアル』を参照してください。

1.4.3 コンパイルからリンクまでの手順

ここでは，ライブラリルーチンを使用しているソースファイルのコンパイルからリンクまでの手順を説明します。

1.4.3.1 コンパイルとアセンブル

ソースファイルのコンパイルとアセンブルについては，ライブラリルーチンを使用しているかどうかを意識する必要はありません。

例

ソースファイル foo.c のコンパイルとアセンブルは，次のように行ないます。

```
CCU8 /TM610001 FOO.C
RASU8 FOO.ASM
```

C ソースレベルデバッグ情報を含んだオブジェクトファイルを作成するには CCU8 に/SD オプション，RASU8 に/SD オプションが必要になります。

1.4.3.2 ライブラリのリンク

コンパイルとアセンブル作業の次は，アブソリュートオブジェクトファイルを作成するために RLU8 を使用してリンク作業を行ないます。このとき，コンパイルとアセンブル作業で作成したオブジェクトファイルの他に，スタートアップルーチンとライブラリファイルを指定してください。

例 1

FOO.OBJ のリンクは，次のように行ないます。

```
RLU8 FOO C:¥U8¥STARTUP¥S610001SW,,, C:¥U8¥LIB¥LU8100SW.LIB /CC
```

この例では，スタートアップルーチン S610001SW.OBJ は，C:¥U8¥STARTUP に置かれています。また，ライブラリファイル LU8100SW.LIB は，C:¥U8¥LIB に置かれています。

環境変数 LIBU8 が示すパスにライブラリファイルを置いている場合は，ライブラリファイルのパスの指定を省略することができます。

例 2

LU8100SW.LIB が C:¥U8¥LIB に置かれており，環境変数 LIBU8 に C:¥U8¥LIB が設定されている場合，RLU8 のコマンドラインは次のようになります。

```
RLU8 FOO C:¥U8¥STARTUP¥S610001SW,,, LU8100SW.LIB /CC
```

1.5 ヘッダファイルの役割

ヘッダファイルは，ライブラリとユーザプログラムのインターフェースの役割を持っています。ライブラリルーチンに対応したヘッダファイルをインクルードすることによって，コンパイラは，ライブラリルーチンの形式（プロトタイプ）や，それらのルーチンが使用する定数や型を知ることができます。

1.5.1 マクロ，定数，型のインクルード

ヘッダファイルのインクルードは，ライブラリに含まれるマクロ，定数，型を定義するために必要です。

ヘッダファイルには，ライブラリルーチンに使用されるマクロ，定数，型の定義が含まれています。プログラマもまた，これらを使用することがあります。ライブラリルーチンが使用するこれらの定義内容と，ユーザプログラムで使用される定義内容は，まったく同じものでなければなりません。

多くの場合，プログラマはヘッダファイルに含まれるマクロ，定数，型の意味さえ知っていればよく，実際の定義内容を知っておく必要はありません。

例

可変引数リストを使用する例です。マクロ va_start，va_arg，va_end および型 va_list は，stdarg.h で定義されていますので，このヘッダファイルをインクルードしています。プログラマは，実際の定義内容を知る必要はありません。

```
#include <stdarg.h>

int func(int num , ...)
{
     int i;
     int total;
     va_list arg;

     va_start(arg , num);
     total = 0;
     for(i = 0 ; i < num ; ++i)
     {
          total += va_arg(arg , int);
     }
     va_end(arg);
     return total;
}
```

1.5.2 関数プロトタイプ宣言のインクルード

ヘッダファイルには，ライブラリのすべての関数の呼び出し形式，すなわち各引数の型と戻り値の型の宣言が含まれています。この宣言は，一般にプロトタイプ宣言と呼ばれています。

コンパイラは，ユーザプログラムで使用されるライブラリ関数の呼び出しの形式，すなわち引数の個数とそれぞれの型，および戻り値の型が，ヘッダファイルのプロトタイプ宣言と一致しているかどうかをチェックします。もし一致していない場合には，コンパイラはワーニングまたはエラーを報告します。

コンパイラによる型チェックは，プログラムの安全性の上できわめて重要です。関数の呼び出し形式の不一致による不具合は，アルゴリズムのミスによる不具合と異なり，発見しにくいからです。

例

strlen 関数を使用する例です。

```
#include <string.h>

int i;

int func(void)
{
     int len;
     .
     .
     .
     len = strlen(i);          /* ワーニング */
     .
     .
     .
}
```

ヘッダファイル string.h の中で，strlen 関数のプロトタイプは次のように宣言されています。

```
     size_t strlen(char *);
```

strlen 関数の呼び出しでは，引数に int 型の変数 i を指定しているので，コンパイラは，この呼び出しに対して，ワーニングを報告します。

コンパイラがこのようなチェックを行なうのは，プログラムの先頭で string.h をインクルードしているからです。もし，string.h をインクルードしていなければ，コンパイラはチェックを行ないません。

1.6 関数とマクロ

1.6.1 マクロと関数の違い

本書において，“ライブラリルーチン”とは，実際には関数とパラメータ付きのマクロの両方を指しています。RTLU8 に含まれるライブラリルーチンは，関数であるか，マクロであるか，もしくは関数とマクロの両方で実装されています。「1.8　各ヘッダファイルの内容」，および第 3 章以降のライブラリリファレンスでは，各ライブラリルーチンがどのように実装されているかを明確に示しています。

通常プログラマは，使用するルーチンが関数なのか，それともマクロなのかを意識する必要はありません。プログラマが，関数とマクロの相違点を意識する必要があるのは，次のような場合です。

（1） 関数の呼び出しはサブルーチンのコールとして展開されますが，マクロの呼び出しは前処理時にインラインコードとして展開されます。すなわち，マクロの呼び出しは，関数の呼び出しによるオーバーヘッドがない分，高速になります。ただし，マクロは，呼び出しの度に同じコードが何回も展開されるため，関数を使用する場合に比べて，プログラムのサイズが大きくなります。

（2） 関数名はコンパイル時にアドレスとしての意味を持っていますが，マクロ名は前処理時に展開され，コンパイル時にはすでに消滅しています。これは，関数へのポインタを経由してマクロを使用することができないことを意味しています。

（3） コンパイラは，関数の呼び出しに対する型チェックを行ないますが，マクロの呼び出しに対しては，型チェックを行ないません。すなわち，マクロの呼び出しの際の引数と戻り値の型については，プログラマ自身が注意しなければなりません。

1.6.2 マクロオーバライドされたルーチンを関数として呼び出す方法

RTLU8 に含まれるライブラリルーチンのいくつかは，関数とマクロの両方で提供されています。ctype.h の中の toupper などは，その一例です。このようなルーチンは，「1.8　各ヘッダファイルの内容」，および第 3 章以降のライブラリリファレンスで，“マクロ・関数”と示しています。

このようなルーチンは，ヘッダファイルの中で最初に関数プロトタイプの宣言があり，その後でマクロとして定義されていますので，通常はマクロが使用されます。ここでは，このようなルーチンを関数として使用する 2 つの方法を紹介します。

1.6.2.1 #undefでマクロを無効にする

1 つめは，#undef 前処理指令を使用して，ルーチン名のマクロとしての定義を無効にする方法です。#undef 前処理指令は，#include 前処理指令によりヘッダファイルをインクルードする行と，最初にルーチンを使用する行の間に記述してください。#include 前処理指令の直後が，最も安全です。

例

touppper マクロを#undef 前処理指令で無効にする例です。

```
#include <ctype.h>
#undef toupper      /* マクロを無効にする */

void func(void)
{
     int c;
     .
     .
     .
     c = toupper(c) ;    /* 関数として呼び出される */
     .
     .
     .
}
```

1.6.2.2 ルーチン名を小カッコで囲む

2 つめは，ライブラリルーチンを呼び出す際に，そのルーチン名を小カッコで囲む方法です。プリプロセッサは，パラメータ付きのマクロをマクロ名の直後に左カッコが存在することを確認してマクロの展開を行ないます。したがって，ルーチン名を小カッコで囲むことによって，プリプロセッサのマクロ展開を阻止することができます。

例

小カッコで toupper を囲むことによって，toupper を関数として呼び出す例です。

```
#include <ctype.h>

void func(void)
{
     int  c;
     .
     .
     .
     c = (toupper)(c) ;  /* 関数として呼び出される */
     .
     .
     .
}
```

1.7 各ヘッダファイルの内容

ここでは，RTLU8 に含まれる関数，マクロ，グローバル変数，定数マクロおよび型を，ヘッダファイル別に説明します。

種別欄には，次のいずれかを示します。

関数

マクロ

マクロ・関数

定数マクロ

型

グローバル変数

マクロ・関数とは，関数とマクロの両方が用意されていることを表わします。関数，マクロ，マクロ・関数の詳細は，第 3 章以降のライブラリリファレンスを参照してください。

1.7.1 文字の分類と変換 <ctype.h>

ctype.h には，1 バイト文字の種類の判定と変換を行なうルーチンが宣言されています。

名前	種別	解説
isalnum	マクロ・関数	文字が数字または英字であるかどうかを調べます。
isalpha	マクロ・関数	文字が英字であるかどうかを調べます。
iscntrl	マクロ・関数	文字が制御文字（0x00～0x1F および 0x7F）かどうかを調べます。
isdigit	マクロ・関数	文字が数字かどうかを調べます。
isgraph	マクロ・関数	文字がスペース（' '）を除く印字可能文字（0x21～0x7E）かどうかを調べます。
islower	マクロ・関数	文字が英小文字かどうかを調べます。
isprint	マクロ・関数	文字がスペース（' '）を含む印字可能文字（0x20～0x7E）かどうかを調べます。
ispunct	マクロ・関数	文字が句切り文字（0x21～0x2F, 0x3A～0x40, 0x5B～0x60, 0x7B～0x7E のいずれか）であるかどうかを調べます。
isspace	マクロ・関数	文字が空白文字（0x09～0x0D および ' '）かどうかを調べます。
isupper	マクロ・関数	文字が英大文字かどうかを調べます。
isxdigit	マクロ・関数	文字が 16 進文字かどうかを調べます。
tolower	マクロ・関数	英大文字を英小文字に変換します。
toupper	マクロ・関数	英小文字を英大文字に変換します。

1.7.2 エラーの識別 <errno.h>

errno.h は，ライブラリルーチン内で発生するエラーに関する情報を含みます。errno.h では，グローバル変数 errno とそれにセットされる定数マクロが定義されています。

名前	種別	解説
errno	グローバル変数	エラー状態を保持する volatile int 型のグローバル変数です。初期値 0 であり，ライブラリルーチン内でエラーが発生したときに，エラーの状態に応じて，以下に示す 0 以外の値をセットします。
EDOM	定数マクロ	定義域エラーを表わす定数です。定義域エラーは，例えば asin 関数に 1 より大きい値や-1 より小さい値を指定する場合のように，数値計算関数が定義域以外の値に対して計算しようとするときに発生します。
ERANGE	定数マクロ	オーバーフローを表わす定数です。オーバーフローは，数値計算関数の計算結果が，double 型で表現できる値の範囲を越えたときに発生します。

1.7.3 浮動小数点の各制限値 <float.h>

float.h では，float 型，double 型，そして long double 型浮動小数点の制限値を表わす定数マクロが定義されています。CCU8 では，long double 型は double 型と同じですので，long double 型の各制限値もまた，double 型のものと同じになります。

名前	種別	解説
DBL_DIG	定数マクロ	double 型の 10 進精度の桁数です。
DBL_EPSILON	定数マクロ	double 型において，1.0+DBL_EPSILON は 1.0 と異なる値であると判定できる最小の正の浮動小数点数です。
DBL_MANT_DIG	定数マクロ	double 型の仮数部のビット数です。
DBL_MAX	定数マクロ	double 型で表現できる最大の値です。
DBL_MAX_EXP	定数マクロ	2 進表現において double 型で表現できる最大の値の指数に 1 を加えた値です。
DBL_MAX_10_EXP	定数マクロ	10 進表現において double 型で表現できる最大の値の指数です。
DBL_MIN	定数マクロ	double 型で表現できる最小の値です。
DBL_MIN_EXP	定数マクロ	2 進表現において double 型で表現できる最小の値の指数に 1 を加えた値です。

名前	種別	解説
DBL_MIN_10_EXP	定数マクロ	10 進表現において double 型で表現できる最小の値の指数です。
FLT_DIG	定数マクロ	float 型の 10 進精度の桁数です。
FLT_EPSILON	定数マクロ	float 型において 1.0+FLT_EPSILON は 1.0 と異なる値であると判定できる最小の正の浮動小数点数です。
FLT_MANT_DIG	定数マクロ	float 型の仮数部のビット数です。
FLT_MAX	定数マクロ	float 型で表現できる最大の値です。
FLT_MAX_EXP	定数マクロ	2 進表現において float 型で表現できる最大の値の指数に 1 を加えた値です。
FLT_MAX_10_EXP	定数マクロ	10 進表現において float 型で表現できる最大の値の指数です。
FLT_MIN	定数マクロ	float 型で表現できる最小の値です。
FLT_MIN_EXP	定数マクロ	2 進表現において float 型で表現できる最小の値の指数に 1 を加えた値です。
FLT_MIN_10_EXP	定数マクロ	10 進表現において float 型で表現できる最小の値の指数です。
FLT_RADIX	定数マクロ	浮動小数点数の指数の底です。
FLT_ROUNDS	定数マクロ	最近値丸めを行なっていることを表わします。
LDBL_DIG	定数マクロ	DBL_DIG と同じです。
LDBL_EPSILON	定数マクロ	DBL_EPSILON と同じです。
LDBL_MANT_DIG	定数マクロ	DBL_MANT_DIG と同じです。
LDBL_MAX	定数マクロ	DBL_MAX と同じです。
LDBL_MAX_EXP	定数マクロ	DBL_MAX_EXP と同じです。
LDBL_MAX_10_EXP	定数マクロ	DBL_MAX_10_EXP と同じです。
LDBL_MIN	定数マクロ	DBL_MAX_10_EXP と同じです。
LDBL_MIN_EXP	定数マクロ	DBL_MIN_EXP と同じです。
LDBL_MIN_10_EXP	定数マクロ	DBL_MIN_10_EXP と同じです。

1.7.4 整数の各制限値 <limits.h>

limits.h では，各整数型の制限値を表わす定数マクロが定義されています。

名前	種別	解説
CHAR_BIT	定数マクロ	8 char 型のビット数です。
CHAR_MAX	定数マクロ	127 char 型の最大値です。
CHAR_MIN	定数マクロ	-128 char 型の最小値です。
INT_MAX	定数マクロ	32767 int 型の最大値です。
INT_MIN	定数マクロ	-32768 int 型の最小値です。
LONG_MAX	定数マクロ	2147483647 long int 型の最小値です。
LONG_MIN	定数マクロ	-2147483648 long int 型の最小値です。
SCHAR_MAX	定数マクロ	127 signed char 型の最大値です。
SCHAR_MIN	定数マクロ	-128 signed char 型の最小値です。
SHRT_MAX	定数マクロ	32767 int 型の最大値です。
SHRT_MIN	定数マクロ	-32768 int 型の最小値です。
UCHAR_MAX	定数マクロ	255 unsigned char 型の最大値です。
UINT_MAX	定数マクロ	65535 unsigned int 型の最大値です。

名前	種別	解説
ULONG_MAX	定数マクロ	4294967295 unsigned long int 型の最大値です。
USHRT_MAX	定数マクロ	65535 unsigned short int 型の最大値です。

1.7.5 数値計算関数 <math.h>

math.h では，各種の数値演算関数が宣言されています。計算は，すべて double 型で行なわれています。いくつかの数値計算関数は，エラーが起こったときに，グローバル変数 errno にエラー値をセットします。詳細は，「第 3 章　標準組み込みルーチンリファレンス」の各ルーチンの説明を参照してください。

名前	種別	解説
HUGE_VAL	定数マクロ	double 型で表現できる最大数です。無限大を表わします。
exp	関数	指数関数を計算します。
frexp	関数	浮動小数点数を仮数部と指数部に分けます。
frexp_n		
frexp_f		
ldexp	関数	引数と 2 のべき乗の積を計算します。
log	マクロ・関数	自然対数を計算します。
log10	マクロ・関数	常用対数を計算します。
modf	関数	浮動小数点数を整数部と小数部とに分けます。
modf_n		
modf_f		
cosh	関数	双曲線コサインを計算します。
sinh	関数	双曲線サインを計算します。
tanh	関数	双曲線タンジェントを計算します。
ceil	関数	浮動小数点数の，整数への切り上げ値を計算します。
fabs	関数	浮動小数点数の絶対値を求めます。
floor	関数	浮動小数点数の，整数への切り下げ値を計算します。
fmod	関数	浮動小数点数の剰余を求めます。
pow	関数	べき乗を計算します。

名前	種別	解説
sqrt	関数	平方根を計算します。
acos	マクロ・関数	逆コサインを計算します。
asin	マクロ・関数	逆サインを計算します。
atan	関数	逆タンジェントを計算します。
atan2	関数	引数どうしの商の逆タンジェントを計算します。atan では計算できないような大きい値の逆タンジェントを求めることができます。
cos	マクロ・関数	コサインを計算します。
sin	マクロ・関数	サインを計算します。
tan	関数	タンジェントを計算します。

1.7.6 グローバルジャンプ <setjmp.h>

setjmp.h には，グローバルジャンプを実現するための関数の宣言，およびマクロ，型の定義が含まれています。これらのルーチンを使用すれば，関数外への分岐が可能になります。

名前	種別	解説
jmp_buf jmp_buf_n jmp_buf_f	型	グローバルジャンプは，setjmp によって環境を保存し，longjmp によってその環境を復帰することで実現されます。jmp_buf は，保存される環境を表わす型です。
setjmp setjmp_n setjmp_f	マクロ	環境を jmp_buf型の引数に保存します。
longjmp longjmp_n longjmp_f	関数	setjmp によって保存された環境を復帰します。その結果，プログラムの実行は，setjmp を呼んだ場所に移ります。

1.7.7 可変引数リスト <stdarg.h>

stdarg.h では，関数の可変引数リストを実現するための定義や宣言を行なっています。これらのルーチンを使用すれば，コンパイラの引数の扱い方を，アセンブリ言語レベルで意識することなく，可変長の引数リストを持つ関数を作成することができます。

名前	種別	解説
va_list	型	可変引数リストに関する情報を保持するデータの型です。va_start，va_arg，va_end の各ルーチンで使用されます。
va_start	マクロ	可変引数リストの参照の準備をします。va_arg を使用する前に，必ずこのルーチンをコールしなければなりません。
va_arg	マクロ	可変引数リストにおける次の引数の値を返します。va_arg を使用することによって，2 番目以降の引数を順番に参照することができます。
va_end	マクロ	可変引数リストの参照の後始末をします。

1.7.8 汎用的な型とマクロ <stddef.h>

stddef.h では，汎用的に使用されるデータ型やマクロが定義されています。

名前	種別	解説
NULL	定数マクロ	null ポインタを表わします。
offsetof	マクロ	構造体のメンバの，構造体の先頭からのバイト数を与えます。
ptrdiff_t	型	符号付きの整数型であり，2 つのポインタの差を表わします。
size_t	型	演算子 sizeof の演算結果を表わす，符号なし整数型です。

1.7.9 入出力関連処理 <stdio.h>

stdio.h では，入出力処理を行なうルーチンの宣言，およびそれらのルーチンが使用するマクロや型の定義が含まれています。

名前	種別	解説
EOF	定数マクロ	-1 ファイルの終了を表します。エラーのときの戻り値としても使用されます。
FILE	型	ストリームの記述の型です。
stderr	マクロ	標準エラーストリームへのポインタです。
stdin	マクロ	標準入力ストリームへのポインタです。
stdout	マクロ	標準出力ストリームへのポインタです。
fflush	関数	ストリームをフラッシュします。
fgetc	関数	ストリームから文字を取得します。
fgets	関数	ストリームから文字列を取得します。
fgets_n		
fgets_f		
fprintf	関数	ストリームに書式化された出力を行ないます。
fprintf_n		
fprintf_f		
fputc	関数	ストリームへ 1 文字を出力します。
fputs	関数	ストリームに文字列を出力します。
fputs_n		
fputs_f		
fread	関数	ストリームからデータを読み込みます。
fread_n		
fread_f		
fscanf	関数	入力ストリームから入力をスキャンし，書式化します。
fscanf_n		
fscanf_f		

名前	種別	解説
fwrite	関数	データをストリームに書き込みます。
fwrite_n		
fwrite_f		
getc	マクロ・関数	ストリームから 1 文字を取得します。
getchar	マクロ・関数	標準入力から 1 文字を取得します。
gets	関数	標準入力から文字列を読込みます。
gets_n		
gets_f		
printf	関数	書式化された出力を標準出力に書き出します。
printf_n		
printf_f		
putc	マクロ・関数	ストリームに 1 文字を出力します。
putchar	マクロ・関数	文字を標準出力に出力します。
puts	関数	標準出力に文字列を出力します。
puts_n		
puts_f		
scanf	関数	標準入力ストリームをスキャンして書式付きで入力します。
scanf_n		
scanf_f		
sprintf	関数	フォーマットされたデータを文字列に書き出します。
sprintf_nn		
sprintf_nf		
sprintf_fn		
sprintf_ff		
sscanf	関数	フォーマットされたデータを文字列から読み込みます。
sscanf_nn		
sscanf_nf		
sscanf_fn		
sscanf_ff		

名前	種別	解説
ungetc	関数	入力ストリームに 1 文字をプッシュバックします。
vprintf	関数	書式付き出力を書き込みます。
vprintf_n		
vprintf_f		
vsprintf	関数	フォーマットされたデータを文字列に書き出します。
vsprintf_nn		
vsprintf_nf		
vsprintf_fn vsprintf_ff		

1.7.10 汎用ユーティリティ <stdlib.h>

stdlib.h では，汎用的に使用できるユーティリティルーチンと，それらのルーチンが使用するマクロや型が定義されています。

名前	種別	解説
div_t	型	関数 div の結果の型で，quot, rem の 2 つの int 型をメンバに持つ構造体です。
ldiv_t	型	関数 ldiv の結果の型で，quot, rem の 2 つの long 型をメンバに持つ構造体です。
RAND_MAX	定数マクロ	32767 関数 rand によって返される疑似乱数の最大値です。
abs	関数	int 型整数値の絶対値を返します。
atof	マクロ・関数	文字列を double 型の浮動小数点数に変換します。
atof_n		
atof_f		
atoi	マクロ・関数	文字列を int 型の整数に変換します。
atoi_n		
atoi_f		
atol	マクロ・関数	文字列を long 型の整数に変換します。
atol_n		
atol_f		

名前	種別	解説
bsearch	関数	ソート済みの配列内から，指定した項目をバイナリサーチします。
bsearch_nn		
bsearch_nf		
bsearch_fn		
bsearch_ff		
calloc	関数	必要な量のメモリを割り当てます。
calloc_n		
calloc_f		
div	関数	2 つの int 型整数値の商と剰余を計算します。div_t 型構造体に商と剰余をセットして，その構造体を返します。
free	関数	メモリを解放します。
free_n		
free_f		
labs	関数	long 型整数値の絶対値を返します。
ldiv	関数	2 つの long 型整数値の商と剰余を計算します。ldiv_t 型構造体に商と剰余をセットして，その構造体を返します。
malloc	関数	メモリを割り当てます。
malloc_n		
malloc_f		
qsort	関数	クイックソートアルゴリズムを使用して，配列の中の要素をソートします。
qsort_n		
qsort_f		
rand	関数	疑似乱数を発生させます。
realloc	関数	メモリの再割り当てをします。
realloc_n		
realloc_f		
srand	マクロ・関数	rand によって与えられる疑似乱数の系列を初期化します。

名前	種別	解説
strtod	マクロ・関数	文字列を double 型の浮動小数点数に変換します。
strtod_n		
strtod_f		
strtol	関数	文字列を long 型の数値に変換します。
strtol_n		
strtol_f		
strtoul	マクロ・関数	文字列を unsigned long 型の数値に変換します。
strtoul_n		
strtoul_f		

1.7.11 文字列操作 <string.h>

string.h では，文字列やメモリ領域を操作する関数が宣言されています。

名前	種別	解説
memchr	関数	メモリ領域内で，ある 1 バイトデータが最初に現れる場所を探します。
memchr_n		
memchr_f		
memcmp	関数	2 つのメモリ領域内を比較します。
memcmp_nn		
memcmp_nf		
memcmp_fn		
memcmp_ff		
memcpy	関数	メモリ領域のデータを，他のメモリ領域へコピーします。
memcpy_nn		
memcpy_nf		
memcpy_fn		
memcpy_ff		

名前	種別	解説
memmove memmove_nn memmove_nf memmove_fn memmove_ff	関数	メモリ領域のデータを，他のメモリ領域へコピーします。memcpy と異なり，メモリ領域が重複していても正常に動作します。
memset memset_n memset_f	関数	一定のメモリ領域を，指定した 1 バイトデータで満たします。
strcat strcat_nn strcat_nf strcat_fn strcat_ff	関数	文字列の連結を行ないます。
strchr strchr_n strchr_f	関数	文字列中で，ある文字が最初に現れる場所を探します。
strcmp strcmp_nn strcmp_nf strcmp_fn strcmp_ff	関数	文字列の比較を行ないます。
strcpy strcpy_nn strcpy_nf strcpy_fn strcpy_ff	関数	文字列のコピーを行ないます。

名前	種別	解説
strcspn	関数	一方の文字列の最初の部分で，他方の文字列に含まれている文字が存在しない部分の長さを返します。
strcspn_nn		
strcspn_nf		
strcspn_fn		
strcspn_ff		
strlen	関数	文字列の長さを返します。
strlen_n		
strlen_f		
strncat	関数	文字列の先頭の n バイトを他の文字列の後に連結します。
strncat_nn		
strncat_nf		
strncat_fn		
strncat_ff		
strncmp	関数	文字列の先頭の n バイトを比較します。
strncmp_nn		
strncmp_nf		
strncmp_fn		
strncmp_ff		
strncpy	関数	文字列の先頭の n バイトを他の領域へコピーします。
strncpy_nn		
strncpy_nf		
strncpy_fn		
strncpy_ff		
strpbrk	関数	一方の文字列に含まれている文字のいずれかが，他方の文字列に最初に現れる場所を探します。
strpbrk_nn		
strpbrk_nf		
strpbrk_fn		
strpbrk_ff		

名前	種別	解説
strrchr	関数	文字列中で，ある文字が最後に現れる場所を探します。
strrchr_n		
strrchr_f		
strspn	関数	一方の文字列の最初の部分で，他方の文字列に含まれている文字だけで構成されている部分の長さを返します。
strspn_nn		
strspn_nf		
strspn_fn		
strspn_ff		
strstr	関数	一方の文字列中から，他方の文字列を探します。
strstr_nn		
strstr_nf		
strstr_fn		
strstr_ff		
strtok	関数	文字列をトークンに切り分けます。
strtok_nn		
strtok_nf		
strtok_fn		
strtok_ff		

1.8 ランタイムライブラリリファレンスの読み方

第 3 章以降では，RTLU8 ランタイムライブラリに含まれているすべてのルーチンを，アルファベット順に解説します。

各ルーチンの解説は，次の形式で行なっています。

ルーチン名

ページの先頭の左にはルーチン名を示します。

種類

ページの先頭の右にはルーチンの種類（関数，マクロ，またはマクロ・関数）を示します。

機能

ルーチンの機能を簡単に解説します。

形式

ルーチンの宣言または定義が入っているヘッダファイルとルーチンのプロトタイプを示します。また，ルーチンの引数の意味を解説します。

解説

ルーチンの機能と使い方を詳しく解説します。

戻り値

ルーチンの返す値を示します。

参照

関連するルーチン名を示します。

プログラム例

ルーチンを実際に使用したプログラムを示します。ここでは，あくまでルーチンの機能を実際のプログラムで示すことを目的としています。必ずしも，実用的なプログラムが示されるとは限りません。

2 データモデル別ライブラリ関数

2.1 データアクセスに対応したライブラリルーチンの存在

CCU8 には，データメモリ空間に関して，デフォルトのアクセス対象を指定する NEAR モデルと FAR モデルの 2 つのデータモデルが存在します。そのため，RTLU8 には，各々のデータアクセスに応じた独自のライブラリ関数が追加されています。それらのライブラリ関数には，データアクセスを決定する修飾子（__near および__far）が指定されています。__near は NEAR データにアクセスするための修飾子で，__far は FAR データにアクセスするための修飾子です。これらの修飾子を直接記述することで，各々のデータアクセスに応じたライブラリ関数を提供してします。

__near および__far 修飾子の詳細については，『CCU8 ランゲージリファレンス』を参照してください。

2.2 ライブラリ関数の使い方

ここでは，各々のデータアクセスに応じたライブラリ関数の使い方を説明します。

2.2.1 一般的なライブラリ関数の使い方

一般的なライブラリ関数の使い方を例にあげます。

例

strcpy(char **string1*, const char **string2*)などのようなポインタを引数に持つ関数で，引数に修飾子__near および__far を指定しない場合について考えてみます。

```
char ndata[128];
const char fdata[] = ”sample”;

void func()
{
     strcpy(ndata, fdata);
}
```

この場合，strcpy の引数は__near または__far 修飾子が指定されていないので，CCU8 のコマンドラインオプション（/near または/far）によって決定されたデータモデルに限定されることになります。データアクセスを指定したい場合（例えば FAR データを NEAR データにコピーしたい場合など），上記のような記述ではそもそも実現できません。

2.2.2 データモデル対応のライブラリ関数の使い方

RTLU8 では 2 つのデータモデルに対処するために，ポインタを引数に持つ ANSI/ISO9899 C 標準のライブラリルーチンに対応したライブラリを用意しています。そのデータモデル対応のライブラリルーチンの使い方を例にあげます。

例

strcpy 関数の正しい使い方，注意の必要な使い方，間違ったキャストの例を示します。ここでは，CCU8 で/near オプションが指定されていることを前提として説明します。

```
#include <string.h>

const char __near nstr[] = "near string";
const char __far fstr[] = "far string";
char __near nbuf[20];
char __far fbuf[20];
char op_buf[20];

void func(void)
{
     .
     .
     .
     /* 正しい使い方 */
     strcpy(nbuf, nstr);
     strcpy_nn(nbuf, nstr);
     strcpy_nf(nbuf, fstr);
     strcpy_fn(fbuf, nstr);
     strcpy_ff(fbuf, fstr);

     strcpy_nf(nbuf, (char __far*)nstr);
     strcpy_fn((char __far *)nbuf, nstr);
     strcpy_ff((char __far *)nbuf, (char __far *)nstr);
     .
     .
     .
     /* 注意の必要な使い方 */
     strcpy_nf(op_buf, fstr);      /* 指定子のないポインタ型は
                                      /near オプションで NEAR に決定 */
     .
     .
     .
     /* 間違ったキャスト */
     strcpy_nn(nbuf, (char __near *)fstr);
     .
     .
     .
}
```

最後のキャストの例は，特に危険です。CCU8 は文法的に問題がないために，このソーステートメントをエラーにはしません。しかし，実際は FAR ポインタを NEAR ポインタにキャストしているので，FAR ポインタの物理セグメントが無視されてしまい，誤動作する可能性もあります。この場合は，

```
strcpy_nf(nbuf, fstr);
```

を使用しなければなりません。NEAR ポインタを FAR ポインタにキャストする場合は，FAR ポインタの物理セグメントに#0 が追加されるだけです。

データモデル対応のライブラリルーチンは，巻末の「付録」を参照してください。

2.3 データモデル対応のルーチン名のルール

ANSI/ISO9899 C 標準のルーチン名と，RTLU8 が独自に用意するデータモデル対応のルーチン名のルールを次に示します。

ANSI/ISO9899 C 標準のルーチン名の後にアンダスコア（_）ではじまるサフィックスを持つルーチンは，引数としてデータモデル（NEAR または FAR）へのポインタを持ちます。サフィックスの種類とその意味は，次のとおりです。

		対象のデータアクセス	
サフィックス	ポインタ引数の数	最初のポインタ引数	2 つめのポインタ引数
_n	1	NEAR	
_f	1	FAR	
_nn	2	NEAR	NEAR
_nf	2	NEAR	FAR
_fn	2	FAR	NEAR
_ff	2	FAR	FAR

ここで上記の例外として，以下の注意があります。

（1） 標準入出力定義である FILE 構造体へのポインタは，NEAR ポインタに限定します。

（2） 可変引数をもつ関数（printf，scanf など）はフォーマット文字列と同じデータモデルとしていますので，これらはポインタ引数の数には含まれていません。

（3） strtod / strtol / strtoul 関数についてはポインタ引数は 2 つ含まれていますが，必要性が低いものとして_nf と_fn の関数は用意していません。これらの関数では，ポインタ引数をすべて NEAR とした_n と，ポインタ引数をすべて FAR とした_f の関数を用意しています。

実際にデータモデルに対応したライブラリ関数の例を示します。

例えば，atol の関数には次のような種類があります。

関数名	ポインタの種類
atol_n(s)	s は NEAR ポインタ
atol_f(s)	s は FAR ポインタ

strcpy の関数には次のような種類があります。

関数名	ポインタの種類
strcpy_nn(s1 , s2)	s1 は NEAR ポインタ，s2 は NEAR ポインタ
strcpy_nf(s1 , s2)	s1 は NEAR ポインタ，s2 は FAR ポインタ
strcpy_fn(s1 , s2)	s1 は FAR ポインタ，s2 は NEAR ポインタ
strcpy_ff(s1 , s2)	s1 は FAR ポインタ，s2 は FAR ポインタ

3 標準組み込みルーチン リファレンス

abs　　関数

機能

int 型整数値の絶対値を返します。

形式

#include <stdlib.h>

int abs(int *n*);

n　　整数

解説

abs は，整数 *n* の絶対値を返します。

戻り値

abs は 0 から 32767 までの範囲の整数を返します。ただし，*n* が-32768 の場合は-32768 を返します。

参照

fabs labs

プログラム例

```
#include <stdlib.h>

void main(void)
{
 int n,res;

 n = -1234;
 res = abs(n);
}
```

acos　　マクロ・関数

機能

アークコサイン（逆余弦）を計算します。

形式

#include <math.h>

double acos(double *x*);

x　　アークコサインを計算する実数の値

解説

acos は，引数 *x* のアークコサインを計算します。引数 *x* の値は，-1 から 1 までの範囲になければなりません。その範囲外の値を引数として与えると，定義域エラーが生じてグローバル変数 errno に EDOM がセットされます。

戻り値

acos は，0 からπラジアンの範囲にある *x* のアークコサインを返します。

参照

asin atan atan2 cos sin tan

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double res;

 x = 0.5;

 res = acos(x);
}
```

asin　　マクロ・関数

機能

アークサイン（逆正弦）を計算します。

形式

#include <math.h>

double asin(double *x*);

x　　アークサインを計算する実数の値

解説

asin は，引数 *x* のアークサインを計算します。引数 *x* の値は，-1 から 1 までの範囲になければなりません。その範囲外の値を引数として与えると，定義域エラーが生じてグローバル変数 errno に EDOM がセットされます。

戻り値

asin は，- π /2 から π /2 ラジアンの範囲にある *x* のアークサインを返します。

参照

acos atan atan2 cos sin tan

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double res;

 x = 0.5;

 res = asin(x);
}
```

atan 関数

機能

アークタンジェント（逆正接）を計算します。

形式

#include <math.h>

double atan(double *x*);

x アークタンジェントを計算する実数の値

解説

atan は，引数 *x* のアークタンジェントを計算します。

戻り値

atan は，-π/2 からπ/2 ラジアンの範囲にある *x* のアークタンジェントを返します。

参照

acos asin atan2 cos sin tan

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double res;

 x = 0.5;

 res = atan(x);
}
```

atan2 関数

機能

y/x のアークタンジェントを計算します。

形式

#include <math.h>

double atan2(double *y*, double *x*);

x, *y* 任意の実数の値

解説

atan2 は，*y/x* のアークタンジェントを計算します。*x* が 0 または，0 に近い場合でも正しい値を返します。*x* および *y* の値が両方とも 0 の場合は 0 を返します。

戻り値

atan2 は，-π からπラジアンの範囲で *y/x* のアークタンジェントを返します。

参照

acos asin atan cos sin tan

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double y;
 double res;

 x = 2.0;
 y = 3.0;

 res = atan2(y, x);
}
```

atof　　マクロ・関数

機能

文字列を double 型の浮動小数点に変換します。

形式

```
#include  <stdlib.h>
double  atof( const char *s );
double  atof_n( const char __near *s );
double  atof_f( const char __far *s )
```

s　　変換する文字列

解説

atof は，引数 *s* の指す文字列を倍精度浮動小数点に変換し，その値を返します。atof は以下の関数呼び出しと同じです。

```
strtod(s, (char *) NULL);
strtod_n(s, (char __near *) NULL);
strtod_f(s, (char __far *) NULL);
```

文字列 *s* は次の形式に沿ったものでなければなりません。

[*white space*] [*sign*] [*digit*] [.] [*digit*] [{e|E} [*sign*] *digit*]

文字列の各部分の説明は以下のとおりです。

記号	意味
[*white space*]	タブまたはスペース（省略可能）
[*sign*]	符号（省略可能）
[*digit*] [.] [*digit*]	小数を表す文字列（省略可能）
[{e|E} [*sign*] *digit*]	指数部を表す文字列（省略可能）

atof は，認識できない文字を読み込んだところで走査をやめます。また，変換した値が double で表現しきれない場合，HUGE_VAL が返され，errno に ERANGE がセットされます。

戻り値

変換された文字列の値を double 型で返します。

参照

atoi atol strtod strtol strtoul

プログラム例

```
#include <stdlib.h>

void main(void)
{
 double res;

 res = atof("1.234e+6");
}
```

atoi　　マクロ・関数

機能

文字列を int 型の整数に変換します。

形式

```
#include <stdlib.h>
int atoi( const char *s );
int atoi_n( const char __near *s );
int atoi_f( const char __far *s );
```

s　　変換する文字列

解説

atoi は，引数 *s* の指す文字列を int 型の整数値に変換し，その値を返します。atoi は以下の関数呼び出しと同じです。

```
(int)strtol(s, (char **) NULL, 10);
(int)strtol_n(s, (char __near * __near *) NULL, 10);
(int)strtol_f(s, (char __far * __far *) NULL, 10);
```

文字列 *s* は次の形式に沿ったものでなければなりません。

[*white space*] [*sign*] [*digit*]

文字列の各部分の説明は以下のとおりです。

記号	意味
[*white space*]	タブまたはスペース（省略可能）
[*sign*]	符号（省略可能）
[*digit*]	整数を表す文字列（省略可能）

atoi は，認識できない文字を読み込んだところで走査をやめます。atoi ではオーバーフローした場合の結果は未定義です。

戻り値

変換された文字列の値を int 型で返します。

参照

atof atol strtod strtol strtoul

プログラム例

```
#include <stdlib.h>

void main(void)
{
 int res;

 res = atoi("32767");
}
```

atol　　マクロ・関数

機能

文字列を long 型の整数に変換します。

形式

#include <stdlib.h>

long atol(const char ∗*s*);

long atol_n(const char __near ∗*s*);

long atol_f(const char __far ∗*s*);

s　　変換する文字列

解説

atol は，引数 *s* の指す文字列を long 型の整数値に変換し，その値を返します。atol は以下の関数呼び出しと同じです。

```
(long)strtol(s, (char **)NULL, 10);
(long)strtol_n(s, (char __near * __near *)NULL, 10);
(long)strtol_f(s, (char __far * __far *)NULL, 10);
```

文字列 *s* は次の形式に沿ったものでなければなりません。

[*white space*] [*sign*] [*digit*]

文字列の各部分の説明は以下のとおりです。

記号	意味
[*white space*]	タブまたはスペース（省略可能）
[*sign*]	符号（省略可能）
[*digit*]	整数を表す文字列（省略可能）

atol は，認識できない文字を読み込んだところで走査をやめます。変換された値が long 型で表現できる範囲にない場合には LONG_MAX または LONG_MIN が返され，errno には ERANGE がセットされます。

戻り値

変換された文字列の値を long 型で返します。

参照

atof atoi strtod strtol strtoul

プログラム例

```
#include <stdlib.h>

void main(void)
{
 long res;

 res = atol("-2147483647");
}
```

bsearch　　関数

機能

ソート済みの配列内から，指定した項目をバイナリサーチします。

形式

```
#include <stdlib.h>
void *bsearch( const void *key, const void *base, size_t nelem,
                    size_t size, int (*cmp )( const void *, const void *) );
void __near *bsearch_nn( const void __near *key, const void __near *base,
                size_t nelem, size_t size, int (*cmp_nn )( const void __near *, const void __near *) );
void __far *bsearch_nf( const void __near *key, const void __far *base, size_t nelem,
                size_t size, int (*cmp_nf )( const void __near *, const void __far *) );
void __near *bsearch_fn( const void __far *key, const void __near *base, size_t nelem,
                size_t size, int (*cmp_fn )( const void __far *, const void __near *) );
void __far *bsearch_ff( const void __far *key, const void __far *base, size_t nelem,
                size_t size, int (*cmp_ff )( const void __far *, const void __far *) );
```

key　　検索キー

base　　検索する配列

nelem　　配列の要素の個数

size　　各要素のサイズを示すバイト数

cmp, cmp_nn, cmp_nf, cmp_fn, cmp_ff　　比較関数へのポインタ

解説

bsearch は，*nelem* 個の要素を持つ配列 *base* から *key* と一致する要素を探し出し，そのアドレスを返します。指定した項目と一致する要素が見つからなかった場合は NULL が返されます。配列の各要素はあらかじめソートされている必要があります。

cmp で指定される関数はユーザが作成する比較関数で，2 つの void 型へのポインタ（void *）を引数とするものでなければなりません。第 1 引数を *elem1*，第 2 引数を *elem2* とすると，この比較関数は，引数に応じて次のとおりに値を返すものでなければなりません。

条件	戻り値
＊elem1 < ＊elem2	負数
＊elem1 == ＊elem2	0
＊elem1 > ＊elem2	正数

戻り値

key と一致した配列の要素へのポインタを返します。一致する要素がなければ NULL を返します。

参照

qsort

プログラム例

```
#include <stdlib.h>

char *array[5];
char a[10] = "apple";
char b[10] = "cherry";
char c[10] = "orange";
char d[10] = "peach";
char e[10] = "pear";
char **curr_ptr;

int compare(char *, char **);

void main(void)
{
array[0] = a;
array[1] = b;
array[2] = c;
array[3] = d;
array[4] = e;

curr_ptr =
(char **)bsearch("peach", array, 5, sizeof(char *), compare );
}

int compare( char *ele1, char **ele2)
{
 return(strcmp(ele1, *ele2));
}
```

calloc　　関数

機能

必要な量のメモリを割り当てます。

形式

#include <stdlib.h>

void ＊calloc(size_t *nelem*, size_t *size*);

void __near ＊calloc_n(size_t *nelem*, size_t *size*);

void __far ＊calloc_f(size_t *nelem*, size_t *size*);

nelem　　各要素の総数

size　　各要素のサイズ

解説

calloc は，*nelem*×*size* バイトのメモリを，ダイナミックセグメント内に割り当てます。割り当てられたメモリの内容はすべて 0 に初期化されます。

戻り値

calloc は，新しく確保されたメモリを指すポインタを返します。要求した分のメモリが確保できなかった場合，または *nelem*，*size* のいずれかが 0 だった場合は，NULL を返します。

参照

free malloc realloc

プログラム例

```
#include <stdlib.h>
#include <string.h>

void main(void)
{
 char *s;

 s = (char *)calloc(10, sizeof(char));
 strcpy(s, ”sample”);
}
```

ceil　　関数

機能

小数点以下を切り上げます。

形式

#include <math.h>

double ceil(double *x*);

x　　浮動小数点の値

解説

ceil は，引数 *x* 以上の整数のうち，最小の整数を見つけます。

戻り値

ceil は，見つけた整数の double 値を返します。

参照

floor fmod

プログラム例

```
#include <math.h>

void main(void)
{
 double num;
 double up;

 num = 12.3;

 up = ceil(num);
}
```

COS　　マクロ・関数

機能

コサイン（余弦）を計算します。

形式

#include <math.h>

double cos(double *x*);

x　　ラジアン単位の角度

解説

cos は，入力値 *x* のコサインを計算します。

戻り値

cos は，-1 から 1 の範囲にある値を返します。

参照

acos asin atan atan2 sin tan

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double res;

 x = 0.5;

 res = cos(x);
}
```

cosh　　関数

機能

ハイパボリックコサイン（双曲線余弦）を計算します。

形式

#include <math.h>

double cosh(double *x*);

x　　ラジアン単位の角度

解説

cosh は，引数 *x* のハイパボリックコサイン$(e^{x}+e^{-x})/2$ を計算します。

戻り値

cosh は，引数 *x* のハイパボリックコサインを返します。計算結果が大きすぎる場合は，HUGE_VAL を返し，グローバル変数 errno に ERANGE がセットされます。

参照

acos asin atan atan2 cos sin sinh tan tanh

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double res;

 x = 0.5;

 res = cosh(x);
}
```

div　　関数

機能

2 つの int 型整数値の商と剰余を計算します。

形式

#include <stdlib.h>

div_t div(int *numer*, int *denom*);

numer　　被除数

denom　　除数

解説

div は，引数 *numer* を *denom* で割り，div_t 型の構造体を返します。型 div_t は qout, rem の 2 つの int 型をメンバに持ち，div は quot に商を，rem に剰余を入れて返します。

戻り値

div は，quot（商）と rem（剰余）をメンバに持つ構造体を返します。

参照

ldiv

プログラム例

```
#include <stdlib.h>

void main(void)
{
 div_t res;
 int num, den;
 int quot, rem;

 num = 32767;
 den = 1000;

 res = div(num, den);
 quot = res.quot;
 rem = res.rem;
}
```

exp　　関数

機能

指数関数 e^x を計算します。

形式

#include <math.h>

double exp(double *x*);

x　　浮動小数点の値

解説

exp は，指数関数 e^x を計算します。

戻り値

exp は，e^x を返します。オーバーフローが起こると HUGE_VAL を，アンダーフローの場合は 0.0 を返し，どちらの場合もグローバル変数 errno に ERANGE がセットされます。

参照

frexp ldexp log log10 pow sqrt

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double res;

 x = 5.5;

 res = exp(x);
}
```

fabs 関数

機能

浮動小数点数の絶対値を計算します。

形式

#include <math.h>

double fabs(double *x*);

x 浮動小数点の値

解説

fabs は，引数 *x* で指定された浮動小数点数の絶対値を計算します。

戻り値

fabs は，引数 *x* の絶対値を返します。

参照

abs labs

プログラム例

```
#include <math.h>

void main(void)
{
 double num;
 double val;

 num = 12.3;

 val = fabs(num);
}
```

floor 関数

機能

小数点以下を切り捨てを行ないます。

形式

#include <math.h>

double floor(double *x*);

x 浮動小数点の値

解説

floor は，引数 *x* を超えない最大の整数を返します。

戻り値

floor は，引数 *x* を超えない最大の整数を浮動小数点で返します。

参照

ceil fmod

プログラム例

```
#include <math.h>

void main(void)
{
 double num;
 double down;

 num = 12.3;

 down = floor(num);
}
```

fmod　　関数

機能

浮動小数点数の剰余を計算します。

形式

#include <math.h>

double fmod(double *x*, double *y*);

x, *y*　　浮動小数点の値

解説

fmod は，*x*=a*y*+f(a は整数, f は *x* と同符号でなおかつ，｜f｜＜｜*y*｜を満たす値)となるような，x を y で割った余り f を計算します。

戻り値

fmod は，浮動小数点数の剰余を返します。*y* が 0 の場合，グローバル変数 errno に EDOM がセットされます。

参照

ceil fabs floor modf

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double y;
 double res;

 x = 7.0;
 y = 2.0;

 res = fmod(x, y);
}
```

free 関数

機能

メモリを解放します。

形式

#include <stdlib.h>

void free(void ∗*ptr*);

void free_n(void __near ∗*ptr*);

void free_f(void __far ∗*ptr*);

ptr 解放するメモリを指すポインタ

解説

free は，calloc，malloc，または realloc によって割り当てられたメモリを解放します。*ptr* は，calloc，malloc，または realloc によって返されたものでなければなりません。それ以外の領域を指すポインタを指定した場合の動作は保証されません。*ptr* に null を指定した場合には何もせずにリターンします。

戻り値

なし

参照

calloc malloc realloc

プログラム例

```
#include <stdlib.h>
#include <string.h>

void main(void)
{
 char *s;
 s = (char *)malloc(10);
 strcpy(s, "sample");
      .
      .
      .
 free(s);
}
```

frexp　　関数

機能

浮動小数点数を仮数部と指数部に分割します。

形式

#include <math.h>

double frexp(double *x*, int ∗*pexp*);

double frexp_n(double *x*, int __near ∗*pexp*);

double frexp_f(double *x*, int __far ∗*pexp*);

x	浮動小数点の値
pexp	指数を格納する領域へのポインタ

解説

frexp は，引数 *x* が $x=m\times2^n$ と等しくなるように，仮数部 m(m の絶対値は 0.5 以上かつ 1.0 未満)と指数部 n に分割します。また，整数値である指数部 n は，*pexp* が指す位置に格納されます。

戻り値

frexp は，仮数部 m の値を返します。

参照

ldexp modf

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double mant;
 int pexp;

 x = 18.4;

 mant = frexp(x, &pexp);
}
```

isalnum ～ isxdigit　　マクロ・関数

機能

文字の種類を判定します。

形式

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

c　　1 バイト文字（0x00～0xff の整数）

解説

これらのルーチンは，文字 *c* の種類を判定して，その判定結果を返します。これらのルーチンは，ASCII 文字セットを想定しています。

c に 0x00～0xff 以外の値を指定した場合の判定結果は不定です。

各ルーチン名と，それらの判定内容は次のとおりです。

ルーチン名	判定内容
isalnum	文字が数字（'0'～'9'）または英字（'a'～'z'または'A'～'Z'）かどうかを調べます。
isalpha	文字が英字（'a'～'z'または'A'～'Z'）かどうかを調べます。
iscntrl	文字が制御文字（0x00～0x1f および 0x7f）かどうかを調べます。
isdigit	文字が数字（'0'～'9'）かどうかを調べます。
isgraph	文字がスペース（' '）を除く印字可能文字（0x21～0x7e）かどうかを調べます。
islower	文字が英小文字（'a'～'z'）かどうかを調べます。
isprint	文字がスペース（' '）を含む印字可能文字（0x20～0x7e）かどうかを調べます。
ispunct	文字が句切り文字（0x21～0x2f，0x3a～0x40，0x5b～0x60，または 0x7b～0x7e）かどうかを調べます。
isspace	文字が空白文字（0x09～0x0d および' '）かどうかを調べます。
isupper	文字が英大文字（'A'～'Z'）かどうかを調べます。
isxdigit	文字が 16 進文字（'0'～'9'，'a'～'f'，または'A'～'F'）かどうかを調べます。

戻り値

判定条件を満たすときは 0 以外，満たさないときは 0 を返します。

c に 0x00～0xff 以外の値を指定した場合の戻り値は不定です。

参照

toupper tolower

プログラム例

```
#include <ctype.h>

void main(void)
{
 int c;
 int     retval1 , retval2 , retval3 , retval4 , retval5;

 /* 'a'～'z'について，種類の判定を行なう。 */

 for (c = 'a'; c <= 'z'; ++c)
     {
      retval1 = isalnum(c);   /* 英字なので TRUE */
      retval2 = islower(c);   /* 小文字なので TRUE */
      retval3 = isupper(c);   /* 大文字ではないので FALSE */
      retval4 = isdigit(c);   /* 数字ではないので FALSE */
      retval5 = isxdigit(c);  /* 'a'～'f'では TRUE，それ以降は FALSE */
     }
}
```

labs　　関数

機能

long 型整数値の絶対値を返します。

形式

#include <stdlib.h>

long labs(long *n*);

n　　整数

解説

labs は，long 型整数 *n* の絶対値を返します。

戻り値

labs は 0 から 2147483647 までの範囲の整数を返します。ただし，*n* が-2147483648 の場合は -2147483648 を返します。

参照

abs fabs

プログラム例

```
#include <stdlib.h>

void main(void)
{
 long n, res;

 n = -123456;
 res = labs(n);
}
```

ldexp　　関数

機能

仮数部と指数部から実数を計算します。

形式

#include <math.h>

double ldexp(double *x*, int *xexp*);

x　　浮動小数点の値

xexp　　整数の指数

解説

ldexp は，$x \times 2^{xexp}$ の値を計算します。

戻り値

ldexp は，計算した値，$x \times 2^{xexp}$ を返します。また，計算結果が大きすぎる場合，グローバル変数 errno に ERANGE がセットされます。

参照

exp frexp modf

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double val;

 x = 4.5;

 val = ldexp(x, 5);
}
```

ldiv 関数

機能

2 つの long 型整数値の商と剰余を計算します。

形式

#include <stdlib.h>

ldiv_t ldiv(long int *numer*, long int *denom*);

numer 被除数

denom 除数

解説

ldiv は，引数 *numer* を *denom* で割り，ldiv_t 型の構造体を返します。型 ldiv_t は qout， rem の 2 つの long 型をメンバに持ち，ldiv は quot に商を，rem に剰余を入れて返します。

戻り値

ldiv は，quot（商）と rem（剰余）をメンバに持つ構造体を返します。

参照

div

プログラム例

```
#include <stdlib.h>

void main(void)
{
 ldiv_t res;
 long num, den;
 long quot, rem;

 num = 165536;
 den = 1000;

 res = ldiv(num, den);
 quot = res.quot;
 rem = res.rem;
}
```

log　　マクロ・関数

機能

x の自然対数を計算します。

形式

#include <math.h>

double log(double *x*);

x　　対数計算の対象となる値

解説

log は，引数 *x* の自然対数を計算します。

戻り値

log は，計算した値，ln(*x*)を返します。引数 *x* が負の場合はグローバル変数 errno に EDOM がセットされます。引数 *x* が 0 の場合は-HUGE_VAL を，数値が大きすぎる場合は HUGE_VAL を返し，どちらの場合も errno に ERANGE がセットされます。

参照

exp log10

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double res;

 x = 10;

 res = log(x);
}
```

log10 マクロ・関数

機能

常用対数を計算します。

形式

#include <math.h>

double log10(double *x*);

x 対数計算の対象となる値

解説

log10 は，*x* の底が 10 の対数を計算します。

戻り値

log10 は，計算した値を返します。引数 *x* が負の場合はグローバル変数 errno に EDOM がセットされます。引数 *x* が 0 の場合は-HUGE_VAL を，数値が大きすぎる場合は HUGE_VAL を返し，どちらの場合も errno に ERANGE がセットされます。

参照

exp log

プログラム例

```
#include <math.h>

void main(void)
{
 double x;
 double res;

 x = 10;

 res = log10(x);
}
```

longjmp　　関数

機能

グローバルジャンプを行ないます。

形式

#include <setjmp.h>

void longjmp(jmp_buf *environment*, int *value*);

void longjmp_n(jmp_buf_n *environment*, int *value*);

void longjmp_f(jmp_buf_f *environment*, int *value*);

environment　実行環境を保存している領域

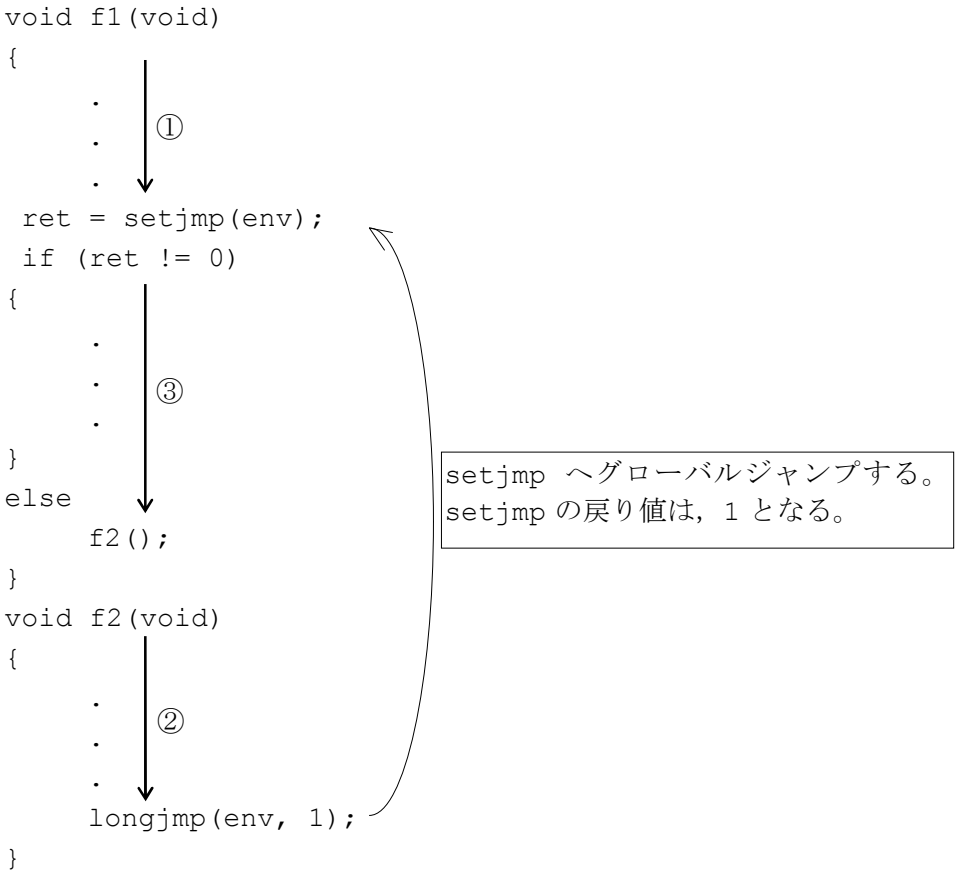
value　　setjmp の戻り値となる値

解説

longjmp は，setjmp 呼び出し位置へのグローバルジャンプを行ないます。

setjmp と longjmp を使用することによって，グローバルなジャンプを行なうことができます。longjmp は，setjmp によりあらかじめ *environment* に保存された実行環境を復帰します。その結果，longjmp を呼び出した後，プログラムは見かけ上 setjmp から戻ったかのように実行されます。*value* は，実行環境復帰時の setjmp の戻り値となります。

次に，setjmp と longjmp の動作を，簡単な例で示します。プログラムは，①，②，③の順番で実行されます。



value の値は，0 以外でなければなりません。*value* に 0 を指定した場合，setjmp は 1 を返します。

longjmp を使用する際には，次の点に注意してください。これらの注意事項に反するプログラムの動作は予測できません。

（1） longjmp を呼び出すより前に，必ず setjmp によって実行環境の保存を行なってください。

（2） setjmp を呼び出した関数がリターンしたあとで，longjmp を呼び出してはなりません。

戻り値

なし

参照

setjmp

プログラム例

```
#include <errno.h>
#include <setjmp.h>

void function1( void );
void function2( void );

jmp_buf environment;

void main(void)
{
     int retval;
     retval = setjmp(environment);
     if ( retval != 0 )
     {
          /* error process  */
     }
function1( );
     .
     .
     .
function2( );
     .
     .
     .
}

void function1( void )
{
     if (errno)
          longjmp(environment , 1);
     .
     .
     .
}

void function2( void )
{
     if (errno)
          longjmp( environment , 2 );
     .
     .
     .
}
```

malloc　　関数

機能

メモリを割り当てます。

形式

```
#include <stdlib.h>
void  *malloc( size_t size );
void  __near *malloc_n( size_t size );
void  __far *malloc_f( size_t size );
```

size　　確保するサイズ

解説

malloc は，*size* バイトのメモリをダイナミックセグメント内に割り当てます。実際には一回の実行につき，メモリ管理とバウンダリの都合上から，*size* が偶数の場合は *size*+2 バイト，*size* が奇数の場合は *size*+3 バイトのメモリが消費されます。割り当てられたメモリの内容は初期化されません。

ダイナミックセグメントとは，RLU8 によってすべての論理セグメントがアドレス空間に割り付けられた後，余った領域の中でもっともサイズの大きい領域に割り当てられるものです。

戻り値

malloc は，割り当てられたメモリを指すポインタを返します。要求したサイズのメモリの割り当てができなかった場合，または *size* が 0 であった場合は，NULL を返します。

参照

calloc free realloc

プログラム例

```
#include <stdlib.h>
#include <string.h>

void main(void)
{
     char *s;

     if ((s = (char *)malloc(10)) != NULL)
     {
          strcpy(s, "sample");
     }
}
```

memchr 関数

機能

一定のメモリ領域内から 1 バイトデータをサーチします。

形式

```
#include <string.h>
void  *memchr( const void *region, int c, size_t count);
void  __near *memchr_n( const void __near *region, int c, size_t count);
void  __far *memchr_f( const void __far *region, int c, size_t count);
```

region　　メモリ領域へのポインタ

c　　サーチするデータ

count　　サーチするバイト数

解説

memchr は，*region* の先頭から *count* バイト内に *c* があるかどうかを調べます。*c* は int 型ですが，その値は 0x00～0xff でなければなりません。

戻り値

region の先頭から *count* バイト内に *c* があった場合には，最初に現れた *c* へのポインタを返します。*c* が見つからなかった場合には，NULL を返します。*count* が 0 の場合も NULL を返します。

参照

memcmp memcpy memset strchr

プログラム例

```
#include <string.h>

char data[16] =
{
     0x00,0x10,0x20,0x30,0x40,0x50,0x60,0x70,
     0x80,0x90,0xa0,0xb0,0xc0,0xd0,0xe0,0xf0
};

void main(void)
{
     char *ptr;
     .
     .
     .
     /* data[8]のアドレスを返す     */
     ptr = memchr( data , 0x80 , 16 );
     .
     .
     .
     /* 0xff がないので NULL を返す     */
     ptr = memchr( data , 0xff , 16 );
     .
     .
     .
     /* 4 バイト目までに 0x80 はないので NULL を返す     */
     ptr = memchr( data , 0x80 , 4 );
     .
     .
     .
}
```

memcmp 関数

機能

2 つのメモリ領域を比較します。

形式

```
#include <string.h>
int  memcmp( const void *region1, const void *region2, size_t count );
int  memcmp_nn( const void __near *region1, const void __near *region2, size_t count );
int  memcmp_nf( const void __near *region1, const void __far *region2, size_t count );
int  memcmp_fn( const void __far *region1, const void __near *region2, size_t count );
int  memcmp_ff( const void __far *region1, const void __far *region2, size_t count );
```

region1　　メモリ領域 1

region2　　メモリ領域 2

count　　比較するバイト数

解説

memcmp は，*region1* と *region2* を，1 バイトごとに先頭から *count* バイト目まで比較します。strcmp と異なり，null 文字（'¥0'）以降も比較の対象となります。

戻り値

比較の結果により，次の値を返します。

戻り値	比較結果
0	*region1* と *region2* は同じ
正値	*region1* は *region2* より大きい
負値	*region1* は *region2* より小さい

参照

memchr memcpy memset strcmp

プログラム例

```
#include <string.h>

char buf1[16] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
char buf2[16] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
char buf3[16] = {0,1,2,3,4,5,6,7,8,9,10, 1, 2, 3, 4, 5};

void main( void )
{
     int     ret;

     /*  同じ内容なので 0 を返す    */
     ret = memcmp(buf1, buf2, 16);
     .
     .
     .
     /*  最初の方が大きいので正値を返す    */
     ret = memcmp(buf1, buf3, 16);
     .
     .
     .
     /*  後の方が大きいので負値を返す    */
     ret = memcmp( buf3 , buf2 , 16 );
     .
     .
     .
}
```

memcpy 関数

機能

あるメモリ領域のデータを他のメモリ領域にコピーします。

形式

#include <string.h>

void ＊memcpy(void ＊*dest*, const void ＊*src*, size_t *count*);

void __near ＊memcpy_nn(void __near ＊*dest*, const void __near ＊*src*, size_t *count*);

void __near ＊memcpy_nf(void __near ＊*dest*, const void __far ＊*src*, size_t *count*);

void __far ＊memcpy_fn(void __far ＊*dest*, const void __near ＊*src*, size_t *count*);

void __far ＊memcpy_ff(void __far ＊*dest*, const void __far ＊*src*, size_t *count*);

dest コピー先

src コピー元

count コピーするバイト数

解説

memcpy は，*src* の *count* バイト目までを，*dest* にコピーします。strcpy や strncpy と異なり，null 文字（'¥0'）以降もコピーの対象となります。

コピー元とコピー先のメモリが重なっているときは，正常な動作は保証できません。重なっている領域間のコピーを行なうためには，memmove を使用してください。

戻り値

dest を返します。

参照

memchr memcmp memmove memset strcpy strncpy

プログラム例

```
#include <string.h>

char data1[16] =
     {
          0x00, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70,
          0x80, 0x90, 0xa0, 0xb0, 0xc0, 0xd0, 0xe0, 0xf0
     };
char data2[16];

void main( void )
{
     char *retptr;
     .
     .
     .
     retptr = memcpy(data2, data1,16);
     .
     .
     .
}
```

memmove 関数

機能

あるメモリ領域のデータを他のメモリ領域にコピーします。

形式

#include <string.h>

void *memmove(void *dest, const void *src, size_t count);

void __near *memmove_nn(void __near *dest, const void __near *src, size_t count);

void __near *memmove_nf(void __near *dest, const void __far *src, size_t count);

void __far *memmove_fn(void __far *dest, const void __near *src, size_t count);

void __far *memmove_ff(void __far *dest, const void __far *src, size_t count);

dest コピー先

src コピー元

count コピーするバイト数

解説

memmove は，*src* の *count* バイト目までを，*dest* にコピーします。コピー元とコピー先のメモリが重なっていても正常に動作します。strcpy や strncpy と異なり，null 文字（'¥0'）以降もコピーの対象となります。

戻り値

dest を返します。

参照

memcpy strcpy strncpy

プログラム例

```
#include <string.h>

char data[] =
{
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f
};

void main(void)
{
      .
      .
      .
      /* data+16 番地以降の 32 バイトを data 番地以降にコピ-する。 */
      /* メモリの重なりがあるが，正常にコピ-される。 */
      memmove(data, data+16 , 32);
      .
      .

}
```

memset 関数

機能

一定のメモリ領域を，特定の 1 バイトデータで初期化します。

形式

#include <string.h>

void ＊memset(void ＊*region*, int *c*, size_t *count*);

void __near ＊memset_n(void __near ＊*region*, int *c*, size_t *count*);

void __far ＊memset_f(void __far ＊*region*, int *c*, size_t *count*);

region メモリ領域

c セットする 1 バイトデータ

count バイト数

解説

memset は，*region* の先頭から *count* バイト目までの各バイトを *c* で初期化します。*c* は int 型ですが，その値は 0x00～0xff でなければなりません。

戻り値

region を返します。

参照

memchr memcpy memcmp memmove

プログラム例

```
#include <string.h>

char data[64];

void main(void)
{
     char *retptr;

     /* バッファ data の先頭から 32 バイトを 0xff で初期化する。 */
     retptr = memset( data , 0xff , 32 );
}
```

modf 関数

機能

浮動小数点数を整数部と小数部に分割します。

形式

#include <math.h>

double modf(double *x*, double ∗*pint*);

double modf_n(double *x*, double __near ∗*pint*);

double modf_f(double *x*, double __far ∗*pint*);

x	浮動小数点の値
pint	整数部を格納する領域へのポインタ

解説

modf は，浮動小数点である引数 *x* を整数部と小数部に分割し，*x* の整数部を *pint* が指す領域に格納し，小数部を関数の値として返します。

戻り値

modf は，引数 *x* の小数部を符号付きで返します。

参照

fmod frexp ldexp

プログラム例

```
#include <math.h>

void main(void)
{
     double x;
     double pint;
     double frac;

     x = 10.2;

     frac = modf(x, &pint);
}
```

offsetof　　マクロ

機能

構造体内でのフィールドのオフセットを求めます。

形式

#include <stddef.h>

size_t offsetof(*structname*, *fieldname*);

structname　　構造体名

fieldname　　構造体のメンバ

解説

offsetof マクロは，フィールド *fieldname* が構造体 *structname* の最初から何バイト目にあるか，そのバイト数を求めます。

戻り値

offsetof マクロは，フィールド *fieldname* が構造体 *structname* の最初から何バイト目にあるか，そのバイト数を返します。

プログラム例

```
#include <stddef.h>

typedef struct {
int member1;
long member2;
char member3;
} structname;

void main(void)
{
size_t ret1;
size_t ret2;
size_t ret3;

ret1 = offsetof(structname, member1);
ret2 = offsetof(structname, member2);
ret3 = offsetof(structname, member3);
}
```

pow　　関数

機能

x の *y* 乗を計算します。

形式

#include <math.h>

double pow(double *x*, double *y*);

x　　数値

y　　*x* のべき乗

解説

pow は，*x* の *y* 乗を計算します。

戻り値

pow は，*x* の *y* 乗を計算した値を返します。引数の値によっては，オーバーフローするか，計算できない場合があります。オーバーフローの場合 pow は HUGE_VAL を返し，グローバル変数 errno に ERANGE がセットされます。また，*x* が負であってかつ *y* が整数でない場合は errno に EDOM がセットされます。引数 *x*，*y* がどちらも 0 であれば，pow は 1 を返します。

参照

exp sqrt

プログラム例

```
#include <math.h>

void main(void)
{
     double x;
     double y;
     double val;

     x = 2.0;
     y = 3.0;

     val = pow(x, y);
}
```

qsort 関数

機能

配列をクイックソートします。

形式

```
#include <stdlib.h>
void qsort( void *base, size_t n, size_t size, int (*cmp)( const void *, const void *) );
void qsort_n( void __near *base, size_t n, size_t size,
              int (*cmp_nn)( const void __near *, const void __near *) );
void qsort_f( void __far *base, size_t n, size_t size,
              int (*cmp_ff)( const void __far *, const void __far *) );
```

base ソートする対象の配列の先頭

n 配列の要素の数

size 各配列の要素のサイズ

cmp, *cmp_nn*, *cmp_ff* 比較関数へのポインタ

解説

qsort は，配列の中の要素をクイックソートします。qsort は，*cmp* が指すユーザ定義の比較関数を呼び出して，配列内の要素をソートします。

cmp で指定される関数はユーザが作成する比較関数で，2 つの void 型へのポインタ（void *）を引数とするものでなければなりません。第 1 引数を *elem1*，第 2 引数を *elem2* とすると，この比較関数は，引数に応じて次のとおりに値を返すものでなければなりません。

条件	戻り値
*elem1 < *elem2	負数
*elem1 == *elem2	0
*elem1 > *elem2	正数

戻り値

なし

参照

bsearch

プログラム例

```
#include <stdlib.h>

int compare(int *, int *);
int base[ ] = {12, 23, 15, 128, 43, 25};

void main( void )
{
     qsort(base, 6, sizeof (int), compare);
}

int compare(int *elem1, int *elem2)
{
     return(*elem1 - *elem2);
}
```

rand 関数

機能

擬似乱数を発生させます。

形式

#include <stdlib.h>

int rand(void);

解説

rand は 0～RAND_MAX の範囲で擬似乱数を発生し，その値を返します。

戻り値

擬似乱数値を返します。

参照

srand

プログラム例

```
#include <stdlib.h>

int random[20];

void main( void )
{
     int i;

     for (i = 0; i < 20; ++i)
     random[i] = rand( );
}
```

realloc 関数

機能

メモリの再割り当てをします。

形式

```
#include <stdlib.h>
void *realloc( void *ptr, size_t size );
void *realloc_n( void __near *ptr, size_t size );
void *realloc_f( void __far *ptr, size_t size );
```

ptr 再割り当ての対象となるメモリを指すポインタ

size 確保するサイズ

解説

realloc は，calloc，または malloc によって割り当てられたメモリの再割り当てをします。

realloc は，要求された大きさのメモリを割り当て，それに対するポインタを返します。新たにメモリが割り当てられた場合は，もとの内容が新たに割り当てられたメモリへコピーされます。*ptr* が NULL であれば，realloc は malloc と同じ動作をします。*size* が 0 で，*ptr* が NULL でない場合は，*ptr* の指すメモリを解放します。

戻り値

realloc は，再割り当てされたメモリを指すポインタを返します。メモリの再割り当てができなかった場合，realloc は NULL を返します。

参照

calloc free malloc

プログラム例

```
#include <stdlib.h>
#include <string.h>

char string1[] = "library";
char string2[] = "reference.";

void main(void)
{
     char *s1, *s2;

     s1 = (char *)malloc(strlen(string1) + 1);
     strcpy(s1, string1);

     /* メモリの再割り当てを行う。
     この時点で s1 の内容が s2 にコピーされている。*/
     s2 = (char *)realloc(s1, strlen(s1) + strlen(string2) + 1);

     /* s2 に string2 を結合する。*/
     strcat(s2, string2);
     /* s2 の内容は"library reference."になる。*/
}
```

setjmp　　マクロ

機能

グローバルジャンプのために，現在のプログラムの実行環境を保存します。

形式

```
#include <setjmp.h>
int   setjmp( jmp_buf environment );
int   setjmp_n( jmp_buf_n environment );
int   setjmp_f( jmp_buf_f environment );
```

environment　　実行環境を保存する領域

解説

setjmp は，現在のプログラムの実行環境を，*environment* に保存します。

setjmp と longjmp を使用することによって，グローバルなジャンプを行なうことができます。setjmp によって保存された実行環境は，longjmp によって復帰されます。その結果，longjmp を呼び出した後，プログラムは見かけ上 setjmp から戻ったかのように実行されます。

setjmp は，環境の保存のための呼び出しでは 0 を返しますが，longjmp による復帰では longjmp の引数 *value* である 0 以外の値を返します。したがって，setjmp の戻り値を参照することによって，現在が環境の保存の直後なのか，longjmp による復帰なのか，さらにどの longjmp からの復帰なのかを知ることができます。

戻り値

実行環境の保存のための呼び出しでは，常に 0 を返します。longjmp の呼び出しの結果として setjmp に戻る場合は，longjmp の第二引数 *value* に指定された 0 以外の値を返します。

参照

longjmp

プログラム例

longjmp を参照してください。

sin　　マクロ・関数

機能

サイン（正弦）を計算します。

形式

#include <math.h>

double sin(double *x*);

x　　ラジアン単位の角度

解説

sin は，引数 *x* のサインを計算します。

戻り値

sin は，引数 *x* のサインを返します。

参照

acos asin atan atan2 cos tan

プログラム例

```
#include <math.h>

void main(void)
{
     double x;
     double res;

     x = 0.5;

     res = sin(x);
}
```

sinh　　関数

機能

ハイパボリックサイン（双曲線正弦）を計算します。

形式

#include <math.h>

double sinh(double *x*);

x　　ラジアン単位の角度

解説

sinh は，引数 *x* のハイパボリックサイン$(e^{x}-e^{-x})/2$ を計算します。

戻り値

sinh は，引数 *x* のハイパボリックサインを返します。計算結果が大きすぎる場合は，適切な符号を持った HUGE_VAL を返し，グローバル変数 errno に ERANGE がセットされます。

参照

acos asin atan atan2 cos cosh sin tan tanh

プログラム例

```
#include <math.h>

void main(void)
{
     double x;
     double res;

     x = 0.5;

     res = sinh(x);
}
```

sprintf　　関数

機能

フォーマットを指定して文字列にテキストを書き込みます。

形式

```
#include <stdio.h>
int sprintf( char *buffer, const char *format [, argument, ...] );
int sprintf_nn( char __near *buffer, const char __near *format [, argument, ...] );
int sprintf_nf( char __near *buffer, const char __far *format [, argument, ...] );
int sprintf_fn( char __far *buffer, const char __near *format [, argument, ...] );
int sprintf_ff( char __far *buffer, const char __far *format [, argument, ...] );
```

buffer	文字列を格納するバッファ
format	フォーマット文字列
argument	変換指定に応じた引数

解説

sprintf は *format* が指すフォーマット文字列にしたがって文字列を作り，*buffer* へ書き込みます。文字列の末尾には null 文字を付加します。

format は，通常の文字と，任意個の変換指定からなる文字列です。*format* の後に続く *argument* の数と各引数の型は，*format* 中の変換指定の数と各変換指定の指定する型に一致していなければなりません。*argument* の数が変換指定の数よりも少ない場合や，変換指定で示される型とそれに対応する引数の型が一致しない場合の動作は保証されません。*argument* の数が変換指定の数よりも多い場合は，その引数は無視されます。

変換指定は次のような形式になっています。

```
% [ flags ] [ width ] [ .prec ] [ {h|l|L} ] type
```

flags にはフラグ文字の並びを指定します。*width* にはフィールド幅を指定します。*.prec* には精度を指定します。h, l および L はサイズ修飾文字です。*type* には変換指定文字を指定します。

フラグ，フィールド幅，精度，サイズ修飾文字はオプションです。オプションの概要を示します。

オプション	内容
flags	出力を左端にそろえるか右端にそろえるか，数値の符号，小数点，8 進数と 16 進数のプレフィクスなどを指定します。
width	出力する文字の最小幅を指定します。
.prec	出力する文字の最大数を指定します。整数の場合は出力する最小桁数を指定します。
{h|l|L}	引数のサイズを決定します。
	h short int
	l long
	L long double

変換指定文字（*type*）

変換指定文字の一覧を以下に示します。

変換指定文字	型	出力の書式
d, i	int	符号付き 10 進の文字列に変換します。
o	unsigned int	符号なし 8 進の文字列に変換します。
u	unsigned int	符号なし 10 進の文字列に変換します。
x	unsigned int	符号なし 16 進の文字列に変換します。10, 11, 12, 13, 14, 15 をそれぞれ a, b, c, d, e, f で表現します。
X	unsigned int	符号なし 16 進の文字列に変換します。10, 11, 12, 13, 14, 15 をそれぞれ A, B, C, D, E, F で表現します。
f	double	符号付きの[-]*d.dddddd* の形式に変換します。
e	double	符号付きの[-]*d.dddddd*e+/-*dd* の形式に変換します。
E	double	e と同じですが，指数部を E で表します。
g	double	e または f の指定する形式に変換します。通常は f の形式で表現されます。指数部が-4 よりも小さいか，精度よりも大きいときには e の形式で表現されます。
G	double	g と同じですが，E または f の指定する形式に変換します。
c	int	1 文字に変換します。
s	char *	精度に達するか，文字列の最後に達するまで，対応する引数が指す文字列中の文字を出力します。
p	void *	入力引数をポインタとして出力します。
n	int *	この時点までに出力された文字数を，引数が指し示す領域に格納します。
%	-	文字%が出力されます。

上表に示された出力の書式は，フラグ文字，幅指定，精度幅，サイズ修飾文字が指定されていないことを前提としています。オプションと変換指定文字の組み合わせによって，出力の書式にどのように影響を与えるかを次ページ以降に示します。

フラグ文字（*flags*）

フラグの種類は次のとおりです。

フラグ文字	意味
-	出力する文字列をフィールド内の左端にそろえます。指定しなければ，文字列を右端にそろえます。
+	数字の先頭に，符号を常に付けるようにします。指定しなければ，値が負のときにのみ符号が出力されます。
スペース（0x20）	値が正の場合，数字の前に空白をおきます。値が負の場合はマイナス記号（-）が付けられます。
#	数値データ型に対応する変換指定文字に適用でき，変換指定文字に応じて適当な書式を割り当てます。次の表を参照してください。
0	変換指定文字 d, e, E, f, g, G, i, u, x, X の前に 0 がついた場合，フィールドをスペース文字の代わりに 0 で埋めます。d, i, o, u, x, X で精度が指定されている場合と，-フラグがある場合には 0 フラグは無視されます。

#が変換指定文字とともに指定されたときは，次のとおりになります。

変換指定文字	#による影響
c, d, i, u, s	影響なし。
o	0 以外の場合は，先頭に 0 を付けます。
x, X	0x（または 0X）を先頭に付けます。
e, E, f	常に小数点を付けます。
g, G	常に小数点を付け，小数点以下の 0 も付けます。

フィールド幅（*width*）

フィールド幅には文字列を書き出すためのフィールドの最小幅を指定します。

フィールド幅の指定がされると，変換された文字列がフィールド幅よりも小さい場合は，その間を埋めるだけのスペース文字を付加します。スペース文字は，-フラグが指定されている場合は右側に，そうでない場合は左側に埋められます。また，フィールド幅の最初の文字が'0'だった場合は，スペース文字の代わりに'0'を付加します。変換された文字列がフィールド幅よりも大きくなった場合，そのフィールド幅は変換された文字列の長さに拡大されます。

フィールド幅を指定するのに，アスタリスク（*）を使って，int 型の引数によりフィールド幅を間接的に指定することもできます。例えば，次のように記述すると，

```
char buf[20];
int width = 8;
int number = 1234;

sprintf(buf, "|%*d|", width, number);
```

buf に出力される文字列は次のようになり，引数 width の値がフィールド幅として用いられます。

```
|    1234|
```

精度（*.prec*）

精度は常にピリオド（.）から始まります。精度の指定の方法は，フィールド幅の指定方法と同じです。ピリオドのみで，後に数字がない場合の精度は 0 とみなされます。

精度を指定した場合に出力される文字数は，各変換指定文字により異なります。精度として *n* を指定した場合，次のとおりになります。

変換指定文字	出力結果
d, i, o, u, x, X	少なくとも *n* 個の数字を出力します。
e, E, f	小数点の後に *n* 個の数字を出力します。
g, G	*n* 個以上の有効数字は出力しません。
s	*n* 個以上の文字は出力しません。

サイズ修飾文字

サイズ修飾文字は，対応する引数の型を変更します。

修飾文字	サイズ
h	d, i, o, u, x, X, n の場合，対応する引数が short int または unsigned short int であると解釈されます。
l	d, i, o, u, x, X, n の場合，対応する引数が long int または unsigned long int であると解釈されます。e, E, f, g, G の場合，対応する引数が double であると解釈されます。
L	e, E, f, g, G の場合，対応する引数が long double であると解釈されます。

重要

可変引数において，アドレスを参照する引数はフォーマット文字列のデータアクセスに依存します。

printf 系関数の可変引数の型は，フォーマット文字列の置かれているメモリ領域にすべて揃えてください。つまり，フォーマット文字列が NEAR データのときはアドレスを参照する引数はすべて NEAR データ，フォーマット文字列が FAR データのときにはすべて FAR データでなければなりません。アドレスを参照する引数に，フォーマット文字列のデータモデルと異なるデータモデルを指定した場合，正常な動作は保証されません。データアクセスの詳細については，『CCU8 ユーザーズマニュアル』を参照してください。以下に，その例を示します。

```
#include <stdio.h>

char __near nbuf[20];
char __far fbuf[20];

char __near nstr[] = "near string";
char __far fstr[] = "far string";

const char __far format[] = "%s %d %p";

int res;

void main( void )
{
     int i = 10;

     strcpy_fn( fbuf, nstr );
     /* format が FAR に置かれているので引数も FAR に揃える */
     res = sprintf_nf( nbuf, format, fbuf, i, fstr );
}
```

戻り値

sprintf は，*buffer* に出力したバイト数を返します。ただし，末尾の null 文字は除きます。エラーが発生すると，sprintf は EOF を返します。

参照

sscanf

プログラム例

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

char buf1[128];
char buf2[128];
char string[20];
int res1;
int res2;

void main(void)
{
     res1 = sprintf(buf1, "|%d|%4x|%04X|%+12.4f|",
                    10, 0xabc, 0xAB, 1234.567);

     strcpy(string, "ABCDEFG");
     res2 = sprintf(buf2, "|%-15s|%15s|", string, "abcdefg");
}
```

sqrt　　関数

機能

実数である引数の平方根を計算します。

形式

#include　　<math.h>

double sqrt(double *x*);

x　　負でない浮動小数点の値

解説

sqrt は，引数 *x* の平方根を計算します。

戻り値

sqrt は，計算した値 *x* の平方根を返します。*x* が負の場合には，グローバル変数 errno に EDOM がセットされ，数値が大きすぎる場合は，errno に ERANGE がセットされます。

参照

exp log pow

プログラム例

```
#include <math.h>

void main(void)
{
     double x;
     double val;

     x = 9.0;

     val = sqrt(x);
}
```

srand　　マクロ・関数

機能

擬似乱数の系列を初期化します。

形式

#include <stdlib.h>

void srand(unsigned int *seed*);

seed　　初期化する値

解説

srand は擬似乱数の系列を初期化します。*seed* の値を変えることにより，rand によって発生する擬似乱数の系列を変えることができます。

戻り値

なし

参照

rand

プログラム例

```
#include <stdlib.h>

int random[20];

void main( void )
{
     int i;

     srand( 123 );
     for (i = 0; i < 20; ++i)
          random[i] = rand( );
}
```

sscanf　　関数

機能

文字列を読み込み，フォーマットにしたがって適当な型に変換します。

形式

```
#include <stdio.h>
int sscanf( const char *string, const char *format [, address, ...] );
int sscanf_nn( const char __near *string, const char __near *format [, address, ...] );
int sscanf_nf( const char __near *string, const char __far *format [, address, ...] );
int sscanf_fn( const char __far *string, const char __near *format [, address, ...] );
int sscanf_ff( const char __far *string, const char __far *format [, address, ...] );
```

string　　読み込む文字列

format　　フォーマット文字列

address　　変換指定に応じた引数

解説

sscanf は，*string* が指す文字列から文字を読み込み，それを *format* で示されるフォーマット文字列にしたがって適切な型のデータに変換し，対応する引数 *address* が指す領域へ格納します。

フォーマット文字列は，空白，変換指定，パーセント（%）以外の文字から構成されます。sscanf は，フォーマット文字列内で空白文字にぶつかると空白以外の文字にぶつかるまで，すべての空白を読み飛ばします。変換指定は，'%'で始まり読み込む文字列の一部をどのように解釈するかを指定するものです。変換指定にぶつかると，sscanf は対応する文字列からトークンを取り出して変換します。それ以外の文字は，読み込む文字列内の文字と一致するあいだ読み飛ばされます。

変換指定の数と，*format* の後に続く引数の数は，同じだけなければなりません。変換指定の数よりも対応する引数の数が少ない場合の動作は保証されません。変換指定の数よりも対応する引数の数が多い場合は，その引数は無視されます。また，変換指定の要求する型と，それに対応する引数の型も一致しなければなりません。一致しない場合，正しい結果は期待できません。

変換指定は次のような形式になっています。

% [*] [*width*] [{h|l|L}] *type*

アスタリスク（*）は，次に続くフィールド（トークン）を読み飛ばすことを示します。対応する引数には何も書き込まれません。*width* には読み込むフィールドの最大文字数（入力幅）を指定します。h, l, および L は引数の型修飾文字で，引数の型を変更するものです。*type* は変換指定文字です。

アスタリスク，入力幅，型修飾文字は，省略可能です。

変換指定文字

変換指定文字の一覧を以下に示します。以下の表は，変換指定文字に対応する引数の型，読み込まれる文字列の解釈のされ方を示しています。

変換指定文字	引数の型	読み込まれる文字列の解釈のされ方
d	int *	文字列を 10 進整数に変換します。文字列の形式は，関数 strtol で基数を 10 に指定したときに解釈される文字列と同じでなければなりません。
i	int *	文字列を 10 進整数に変換します。文字列の形式は，関数 strtol で基数を 0 に指定したときに解釈される文字列と同じでなければなりません。
o	unsigned int *	文字列を 8 進整数に変換します。文字列の形式は，関数 strtol で基数を 8 に指定したときに解釈される文字列と同じでなければなりません。
u	unsigned int *	文字列を符号なし 10 進整数に変換します。文字列の形式は，関数 strtoul で基数を 10 に指定したときに解釈される文字列と同じでなければなりません。
x, X	unsigned int *	文字列を符号なし 16 進整数に変換します。文字列の形式は，関数 strtol で基数を 16 に指定したときに解釈される文字列と同じでなければなりません。
f	float *	文字列を浮動小数点数に変換します。文字列の形式は，関数 strtod で 10 進表記を浮動小数点数に変換するときに解釈される文字列と同じでなければなりません。
e, E	float *	文字列を浮動小数点数に変換します。文字列の形式は，関数 strtod で指数表記を浮動小数点数に変換するときに解釈される文字列と同じでなければなりません。

変換指定文字	引数の型	読み込まれる文字列の解釈のされ方
g, G	float *	文字列を浮動小数点数に変換します。文字列の形式は，関数 strtod で 10 進表記または指数表記を浮動小数点数に変換するときに解釈される文字列と同じでなければなりません。
c	char *	入力幅で指定された数だけの文字を，引数の指す配列へコピーします（空白文字も含まれます）。このとき，コピー先の配列には null 文字はセットされません。入力幅が指定されなかった場合は，1 文字だけが読み込まれます。
s	char *	空白文字を含まない文字列を引数の指す文字列へコピーします。文字列の最後には null 文字がセットされます。
p	void *	文字列を void 型へのポインタとして読み込みます。
n	int *	この時点までに読み込まれた文字数を，引数が指し示す領域に格納します。
%	-	文字%が読み込まれます。引数には何もセットされません。
[...]	char *	[]で囲まれた集合文字列のいずれかに一致する文字を，引数が指す文字列へコピーします。集合文字列には空白文字も含まれます。[]...]となっている場合，']'もスキャンの対象となります。
[^...]	char *	[]で囲まれた集合文字列にいずれも一致しない文字を，引数が指す文字列へコピーします。集合文字列には空白文字も含まれます。[^]...]となっている場合，']'もスキャンの対象となります。

引数の型は，型修飾文字が指定されていない場合を前提としています。型修飾文字が指定されたときにどのような型に変更されるのかを次に示します。

型修飾文字

型修飾文字は，対応する引数の型を変更します。

型修飾文字	解釈される型
h	d, i, o, u, x, X, n の場合，対応する引数は short int または unsigned short int へのポインタであると解釈されます。それ以外の場合は無視されます。
l	d, i, o, u, x, X, n の場合，対応する引数は long int または unsigned long int へのポインタであると解釈されます。e, E, f, g, G の場合，対応する引数は double へのポインタであると解釈されます。それ以外の場合は無視されます。
L	e, E, f, g, G の場合，対応する引数は long double へのポインタであると解釈されます。それ以外の場合は無視されます。

重要

可変引数において，アドレスを参照する引数はフォーマット文字列のデータアクセスに依存します。

scanf 系関数の可変引数の型はフォーマット文字列の置かれているメモリ領域にすべて揃えてください。つまり，フォーマット文字列が NEAR データのときはアドレスを参照する引数はすべて NEAR データ，フォーマット文字列が FAR データのときにはすべて FAR データでなければなりません。アドレスを参照する引数に，フォーマット文字列のデータモデルと異なるデータモデルを指定した場合，正常な動作は保証されません。データアクセスの詳細については，『CCU8 ユーザーズマニュアル』を参照してください。以下に，その例を示します

```
#include <stdio.h>

char __near nstr[] = "input_data 1.234";
const char __far format[] = "%s %lf%n";

char __far fbuf[30];
double __far fd;
int __far fcnt;

int res;

void main( void )
{
     /* format が FAR に置かれているので引数も FAR に揃える */
     res = sscanf_nf( nstr, format, fbuf, &fd, &fcnt );
}
```

戻り値

sscanf は，正しく読み込んだ入力データの個数を返します。エラーが発生した場合には EOF が返されます。

参照

sprintf

プログラム例

```
#include <stdio.h>

int year, month, date;
char name[15];
float height;
int res;

void main(void)
{
     res=sscanf("1993.11.17,T.YAMADA,170.5","%d.%d.%d , %s , %f",
                    &year, &month, &date, name, &height );
}
```

strcat 関数

機能

文字列を結合します。

形式

```
#include <string.h>
char *strcat( char *string1, const char *string2 );
char __near *strcat_nn( char __near *string1, const char __near *string2 );
char __near *strcat_nf( char __near *string1, const char __far *string2 );
char __far *strcat_fn( char __far *string1, const char __near *string2 );
char __far *strcat_ff( char __far *string1, const char __far *string2 );
```

string1　　結合先の文字列

string2　　結合する文字列

解説

strcat は，*string1* の null 文字（'¥0'）以降に，*string2* を結合して，その終端に null 文字（'¥0'）を付加します。

戻り値

string1 を返します。

参照

strncat strcpy strncpy

プログラム例

```
     #include <string.h>

     char string1[128] = "library ";
     char string2[128] = "reference ";

     void main(void)
     {
          char  *retptr;
               .
               .
               .
          /* 文字列"library reference "を生成する。 */
          retptr = strcat(string1 , string2);

          /* 文字列"library reference manual"を生成する。 */
          retptr = strcat( retptr , "manual" );
               .
               .
               .
}
```

strchr 関数

機能

文字列中から，ある文字が最初に現れる位置を求めます。

形式

```
#include <string.h>
char *strchr( const char *string, int c );
char __near *strchr_n( const char __near *string, int c );
char __far *strchr_f( const char __far *string, int c );
```

string 文字列

c 検索する文字

解説

strchr は，*string* 中から *c* を探します。*c* には，null 文字（'¥0'）を指定することもできます。*c* は int 型ですが，その値は 0x00～0xff でなければなりません。

最後に現れる *c* の位置を求める場合は，strrchr を使用してください。

戻り値

文字が最初に現れる位置へのポインタを返します。文字が見つからなかった場合は，NULL を返します。

参照

memchr strcspn strrchr strspn

プログラム例

```
#include <string.h>

char string[] = "01234567890123456789012345678 9";

void main(void)
{
     char *ptr;

     /* 最初に現れる'9'は 9 番目なので，string[9]へのポインタを返す。 */
     ptr = strchr(string , '9');
          .
          .
          .
     /* '¥0'を指定した場合，文字列の終端へのポインタを返す。 */
     ptr = strchr(string , '¥0');
          .
          .
          .
     /* 'A'は文字列中に存在しないので，NULL を返す。 */
     ptr = strchr(string , 'A');
}
```

strcmp 関数

機能

2 つの文字列を比較します。

形式

```
#include  <string.h>
int  strcmp( const char *string1, const char *string2 );
int  strcmp_nn( const char __near *string1, const char __near *string2 );
int  strcmp_nf( const char __near *string1, const char __far *string2 );
int  strcmp_fn( const char __far *string1, const char __near *string2 );
int  strcmp_ff( const char __far *string1, const char __far *string2 );
```

string1　　比較する文字列

string2　　比較する文字列

解説

strcmp は，*string1* と *string2* を辞書形式で比較します。

戻り値

比較の結果により，次の値を返します。

戻り値	比較結果
0	*string1* と *string2* は同じ
正値	*string1* は *string2* より大きい
負値	*string1* は *string2* より小さい

参照

memcmp strncmp

プログラム例

```
#include <string.h>

/* string1 は string2 より大きい。 */
char string1[] = "ABCDE";
char string2[] = "AAAAA";

void main(void)
{
     int retval;

     /* 1 つめの文字列が大きいので，正値を返す。   */
     retval = strcmp(string1, string2);
          .
          .
          .
     /* 2 つめの文字列が大きいので，負値を返す。   */
     retval = strcmp(string2, string1);
          .
          .
          .
     /* 同じ文字列なので，0 を返す。   */
     retval = strcmp(string1, string1);
}
```

strcpy 関数

機能

文字列のコピーを行ないます。

形式

#include <string.h>

char ＊strcpy(char ＊*string1*, const char ＊*string2*);

char __near ＊strcpy_nn(char __near ＊*string1*, const char __near ＊*string2*);

char __near ＊strcpy_nf(char __near ＊*string1*, const char __far ＊*string2*);

char __far ＊strcpy_fn(char __far ＊*string1*, const char __near ＊*string2*);

char __far ＊strcpy_ff(char __far ＊*string1*, const char __far ＊*string2*);

string1 コピー先

string2 コピー元の文字列

解説

strcpy は，*string2* を終端の null 文字（'¥0'）を含めて，*string1* へコピーします。

戻り値

string1 を返します。

参照

memcpy strcat strncat strncpy

プログラム例

```
#include <string.h>

char string[128];

void main(void)
{
     char *retptr;

     retptr = strcpy(string , "string data");
}
```

strcspn 関数

機能

特定の文字群の文字を含まない部分の長さを得ます。

形式

```
#include <string.h>
size_t strcspn( const char *string1, const char *string2 );
size_t strcspn_nn( const char __near *string1, const char __near *string2 );
size_t strcspn_nf( const char __near *string1, const char __far *string2 );
size_t strcspn_fn( const char __far *string1, const char __near *string2 );
size_t strcspn_ff( const char __far *string1, const char __far *string2 );
```

string1 文字列

string2 文字群を内容とする文字列

解説

strcspn は，*string2* に含まれる文字が *string1* 中で最初に現れる場所を探し，*string1* の先頭からのオフセットとして返します。言い換えれば，*string2* に含まれない文字で構成される *string1* の先頭からの部分文字列の長さを求めます。*string1* の終端の null 文字（'¥0'）は，検索の対象とはなりません。

strcspn は，strpbrk とよく似ています。しかし，strpbrk は最初に現れる文字の位置へのポインタを返す点が異なります。また，これらとまったく逆の機能を持つ関数として，strspn が用意されています。

戻り値

string1 の先頭から，*string2* に含まれる文字が最初に現れる位置までの長さを返します。*string2* に含まれる文字が 1 つも存在しないとき，または *string2* が null 文字列（""）のとき，*string1* の長さを返します。

参照

strchr strrchr strpbrk strspn

プログラム例

```
#include <string.h>

char string1[] = "ABCDEFG1234567";
char string2[] = "1234567";

void main(void)
{
     size_t retval;
     .
     .
     .
/*
"ABCDEFG1234567"の中で，"1234567"の中のいずれかの文字が現れるまでに
7 文字存在するので，7 を返す。
*/
     retval = strcspn(string1 , string2);
     .
     .
     .
/*
"ABCDEFG1234567"の中に，"XYZ"のいずれも存在しないので,
文字列の長さを返す。
*/
     retval = strcspn(string1 , "XYZ");
}
```

strlen 関数

機能

文字列の長さを返します。

形式

#include <string.h>

size_t strlen(const char ∗*string*);

size_t strlen_n(const char __near ∗*string*);

size_t strlen_f(const char __far ∗*string*);

string 文字列

解説

strlen は，*string* の長さ，すなわち *string* の先頭から，終端の null 文字（'¥0'）の直前までの文字数（バイト数）を求めます。

戻り値

string の長さを返します。

参照

なし

プログラム例

```
#include <string.h>

char string[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

void main( void )
{
     size_t length;

/* 文字列の長さである 26 を返す。 */
length = strlen(string);
}
```

strncat 関数

機能

文字列の先頭からの一部を，他の文字列の後ろに結合します。

形式

```
#include <string.h>
char *strncat( char *string1, const char *string2, size_t count );
char __near *strncat_nn( char __near *string1, const char __near *string2, size_t count );
char __near *strncat_nf( char __near *string1, const char __far *string2, size_t count );
char __far *strncat_fn( char __far *string1, const char __near *string2, size_t count );
char __far *strncat_ff( char __far *string1, const char __far *string2, size_t count );
```

string1　　結合先の文字列

string2　　結合する文字列

count　　結合する文字数

解説

strncat は，*string1* の null 文字（'¥0'）以降に，*string2* の先頭からの *count* バイトを結合して，その終端に null 文字（'¥0'）を付加します。

count が *string2* の長さより大きいときは，*string2* のすべてが *string1* に連結されます。この動作は，strcat と同じです。*count* が 0 の場合や *string2* が null 文字列（""）の場合には，*string1* の内容は変化しません 。

戻り値

string1 を返します。

参照

strcat strcmp strcpy strncpy

プログラム例

```
#include <string.h>

char string1[128] = "library ";
char string2[128] = "reference";
char string3[128] = "manual";

void main(void)
{
     char *retptr;

     /*
     "reference"の先頭から 3 文字を連結する。
     文字列の内容は，"library ref"となる。
     */
     retptr = strncat(string1, string2 , 3);

     /*
     "manual"の長さより大きい値を指定する。
     文字列の内容は，"library refmanual"となる。
     */
     retptr = strncat(retptr, string3, 20);

     /*
     文字数として 0 を指定する。文字列の内容は，変化しない。
     */
     retptr = strncat(retptr, string3, 0);
}
```

strncmp 関数

機能

2 つの文字列を指定の文字数だけ比較します。

形式

```
#include <string.h>
int strncmp( const char *string1, const char *string2, size_t count );
int strncmp_nn( const char __near *string1, const char __near *string2, size_t count );
int strncmp_nf( const char __near *string1, const char __far *string2, size_t count );
int strncmp_fn( const char __far *string1, const char __near *string2, size_t count );
int strncmp_ff( const char __far *string1, const char __far *string2, size_t count );
```

string1 比較する文字列

string2 比較する文字列

count 比較する文字数

解説

strncmp は，*string1* と *string2* の先頭からの *count* 文字を，辞書形式で比較します。*count* が，比較する文字列の長さより小さいときは，文字列の先頭から *count* バイト目までが，比較の対象となります。*count* が，比較する文字列の長さより大きいときは，終端の null 文字（'¥0'）までが，比較の対象となります。*count* の値が，*string1* と *string2* のどちらの長さよりも大きいときの結果は，strcmp と同じになります。

戻り値

比較の結果により，次の値を返します。

戻り値	比較結果
0	*string1* と *string2* は同じ
正値	*string1* は *string2* より大きい
負値	*string1* は *string2* より小さい

参照

memcmp strcat strcmp strcpy strncat strncpy

プログラム例

```
#include <string.h>

/* string1 は，7 バイト目以降が string2 より大きい。 */
char string1[] = "1234567890";
char string2[] = "1234560000";

void main(void)
{
     int retval;

     /* 6 バイト目までの比較。0 を返す。 */
     retval = strncmp(string1 , string2 , 6);

     /* 7 バイト目までの比較。1 つめの文字列が大きいので正値を返す。 */
     retval = strncmp(string1 , string2 , 7);
}
```

strncpy　　関数

機能

指定バイト数だけ，文字列のコピーを行ないます。

形式

```
#include <string.h>
char *strncpy( char *string1, const char *string2, size_t count );
char __near *strncpy_nn( char __near *string1, const char __near *string2, size_t count );
char __near *strncpy_nf( char __near *string1, const char __far *string2, size_t count );
char __far *strncpy_fn( char __far *string1, const char __near *string2, size_t count );
char __far *strncpy_ff( char __far *string1, const char __far *string2, size_t count );
```

string1　　コピー先

string2　　コピー元の文字列

count　　コピーする文字数

解説

strncpy は，*string2* の先頭から *count* バイトを，*string1* へコピーします。*count* が *string2* の長さと等しいか，または小さい場合，コピーした文字列の終わりに null 文字（'¥0'）は付加しません。*count* が *string2* の長さより大きい場合，*string1* には *string2* の全部がコピーされ，さらに残りの *count* バイト目までは，null 文字がセットされます。

戻り値

string1 を返します。

参照

memcpy strcat strncat strcpy

プログラム例

```
#include <string.h>

char string1[] = "string";

char string2[128];

void main(void)
{
     char *retptr;
     .
     .
     .
/*
長さ 6 の文字列，指定文字数 3 の場合。
先頭の 3 文字だけがコピーされる。null 文字は付加されない。
*/
     retptr = strncpy(string2, string1, 3);
     .
     .
     .
/*
長さ 6 の文字列，指定文字数 10 の場合。
"string"がコピされた後，残りの 4 バイトには null 文字がセットされる。
コピー結果は，"string¥0¥0¥0¥0"である。
*/
     retptr = strncpy(string2, string1, 10);
}
```

strpbrk 関数

機能

文字列の中から，特定の文字群に含まれる文字が最初に現れる場所を探します。

形式

```
#include <string.h>
char *strpbrk( const char *string1, const char *string2 );
char __near *strpbrk_nn( const char __near *string1, const char __near *string2 );
char __near *strpbrk_nf( const char __near *string1, const char __far *string2 );
char __far *strpbrk_fn( const char __far *string1, const char __near *string2 );
char __far *strpbrk_ff( const char __far *string1, const char __far *string2 );
```

string1　　文字列

string2　　文字群を内容とする文字列

解説

strpbrk は，*string2* に含まれる文字が *string1* 中で最初に現れる場所を探し，そのポインタを返します。*string1* の終端の null 文字（'¥0'）は，検索の対象とはなりません。

これらのルーチンは，strcspn とよく似ています。しかし，strcspn は最初に現れる文字の先頭からのオフセットを返す点が異なります。

戻り値

string2 に含まれる文字が最初に *string1* に現れる位置へのポインタを返します。*string1* 中に，*string2* に含まれる文字が 1 つも存在しないとき，および *string1* または *string2* が null 文字列（""）のとき，これらのルーチンは NULL を返します。

参照

strchr strcspn strrchr strspn

プログラム例

```
#include <string.h>

char string1[] = "ABCDEFG1234567";
char string2[] = "1234567";

void main(void)
{
     char *ptr;

/*
"ABCDEFG1234567"の中で，"1234567"の中のいずれかの文字は，
7 文字目に最初に現れるので 7 バイト目のポインタを返す。
*/
     ptr = strpbrk(string1, string2);

/*
"ABCDEFG1234567"の中に，"XYZ"のいずれも存在しないので，NULL を返す。
*/
     ptr = strpbrk(string1, "XYZ");

/*
null 文字列を指定した場合は NULL を返す。
*/
     ptr = strpbrk(string1, "");
}
```

strrchr 関数

機能

文字列中から，ある文字が最後に現れる位置を求めます。

形式

```
#include <string.h>
char *strrchr( const char *string, int c );
char __near *strrchr_n( const char __near *string, int c );
char __far *strrchr_f( const char __far *string, int c );
```

string 文字列

c 検索する文字

解説

strrchr は，*string* 中で最後に現れる *c* の位置を求めます。*c* には，null 文字（'¥0'）を指定することもできます。*c* は int 型ですが，その値は 0x00～0xff でなければなりません。最初に現れる *c* の位置を求める場合は，strchr を使用してください。

戻り値

文字が最後に現れる位置へのポインタを返します。文字が見つからなかった場合は，NULL を返します。

参照

memchr strcspn strchr strspn

プログラム例

```
     #include <string.h>

     char string[] = "01234567890123456789012345678 9";

     void main(void)
     {
          char *ptr;

     /* 最後に現れる'0'は 20 番目なので，  */
     /* string[20] へのポインタを返す。  */
          ptr = strrchr(string, '0');
          .
          .
          .
     /* '¥0'を指定した場合，文字列の終端へのポインタを返す。  */
          ptr = strrchr(string, '¥0');
          .
          .
          .
     /* 'A'は文字列中に存在しないので，NULL を返す。   */
          ptr = strrchr(string, 'A');
}
```

strspn 関数

機能

文字列の先頭で，特定の文字群の文字を含む部分の長さを得ます。

形式

```
#include <string.h>
size_t strspn( const char *string1, const char *string2 );
size_t strspn_nn( const char __near *string1, const char __near *string2 );
size_t strspn_nf( const char __near *string1, const char __far *string2 );
size_t strspn_fn( const char __far *string1, const char __near *string2 );
size_t strspn_ff( const char __far *string1, const char __far *string2 );
```

string1 文字列

string2 文字群を内容とする文字列

解説

strspn は，*string2* に含まれない文字が *string1* 中で最初に現れる場所を探し，*string1* の先頭からのオフセットとして返します。言い換えれば，*string2* に含まれる文字だけで構成される *string1* の先頭からの部分文字列の長さを求めます。*string1* の終端の null 文字（'¥0'）は，検索の対象とはなりません。

これとまったく逆の機能を持つ関数として，strcspn が用意されています。

戻り値

string1 の先頭から，*string2* に含まれない文字が最初に現れる位置までの長さを返します。*string1* の 1 文字目が *string2* に含まれる文字ではない場合，および *string1* または string2 が null 文字列（""）の場合は，0 を返します。

参照

strchr strrchr strpbrk strcspn

プログラム例

```
     #include <string.h>

     char string1[] = "ABCDEFGABCDEFG1234567";
     char string2[] = "GFEDCBA";

     void main(void)
     {
          size_t retval;

     /*
     "ABCDEFGABCDEFG1234567"の中で，先頭から 14 文字目までは"GFEDCBA"の中
     文字で構成されているので，14 を返す。
     */
          retval = strspn(string1, string2);

     /*
     "ABCDEFGABCDEFG1234567"の先頭には，"XYZ"の中の文字は無いので 0 を返す。
     */
          retval = strspn(string1, "XYZ");
}
```

strstr　　関数

機能

文字列の中から，部分文字列を探します。

形式

```
#include <string.h>
char *strstr( const char *string1, const char *string2 );
char __near *strstr_nn( const char __near *string1, const char __near *string2 );
char __near *strstr_nf( const char __near *string1, const char __far *string2 );
char __far *strstr_fn( const char __far *string1, const char __near *string2 );
char __far *strstr_ff( const char __far *string1, const char __far *string2 );
```

string1　　サーチされる文字列

string2　　サーチする文字列

解説

strstr は，*string1* の中から *string2* をサーチします。

戻り値

string2 が最初に *string1* 内に現れる位置へのポインタを返します。*string2* が *string1* に存在しない場合，および *string1* が null 文字列（""）の場合は，NULL を返します。また *string2* が null 文字列の場合，*string1* を返します。

参照

strcspn strspn strchr strrchr strpbrk

プログラム例

```
#include <string.h>

char string[] =
/*
0   ---   1   ---   2   ---   3   ---   4
01234567890123456789012345678901234567890
*/
"WORD1     WORD2     WORD3     WORD4    ";

void main(void)
{
     char *ptr;

/* "WORD1"をサーチ。
   string+0 を返す。 */
     ptr = strstr(string, "WORD1");

/* "WORD2"をサーチ。
   string+10 を返す。 */
     ptr = strstr(string, "WORD2");

/* "WORD3"をサーチ。
   string+20 を返す。 */
     ptr = strstr(string, "WORD3");

/* "NOTHING"をサーチ。
   存在しないので，NULL を返す。 */
     ptr = strstr(string , "NOTHING");
}
```

strtod　　マクロ・関数

機能

文字列を double 型の浮動小数点数に変換します。

形式

```
#include <stdlib.h>
double strtod( const char *s, char **endptr );
double strtod_n( const char __near *s, char __near * __near *endptr );
double strtod_f( const char __far *s, char __far * __far *endptr );
```

s　　変換する文字列

endptr　　走査するのを中止した文字を指すポインタ

解説

strtod は，引数 *s* の指す文字列を倍精度浮動小数点数に変換し，その値を返します。文字列 *s* は次の形式に沿ったものでなければなりません。

[*white space*] [*sign*] [*digit*] [.] [*digit*] [{e|E} [*sign*] *digit*]

文字列の各部分の説明は以下のとおりです。

記号	意味
[*white space*]	タブまたはスペース（省略可能）
[*sign*]	符号（省略可能）
[*digit*] [.] [*digit*]	小数を表す文字列（省略可能）
[{e|E} [*sign*] *digit*]	指数部を表す文字列（省略可能）

strtod は，認識できない文字を読み込んだところで走査をやめ，*endptr* が NULL でなければ，その文字の位置を示すポインタを *endptr* にセットします。また，変換した値が double で表現しきれない場合，HUGE_VAL が返され，errno に ERANGE がセットされます。

戻り値

変換された文字列の値を double 型で返します。

参照

atof atoi atol strtol strtoul

プログラム例

```
#include <stdlib.h>

void main(void)
{
     double res;
     char *endp;

     res = strtod("1.234e+6", &endp);
}
```

strtok　　関数

機能

文字列をデリミタによってトークンに分割して，各トークンを順番に返します。

形式

```
#include <string.h>
char *strtok( char *string1, const char *string2 );
char __near *strtok_nn( char __near *string1, const char __near *string2 );
char __near *strtok_nf( char __near *string1, const char __far *string2 );
char __far *strtok_fn( char __far *string1, const char __near *string2 );
char __far *strtok_ff( char __far *string1, const char __far *string2 );
```

string1　　トークンに分割する文字列，または NULL

string2　　デリミタで構成される文字列

解説

ここで言う“トークン”とは，*string2* に含まれる文字以外で構成される *string1* 中の部分文字列です。“デリミタ”とは，*string2* に含まれる文字のことです。例えば，デリミタをスペース（' '），コロン（':'），ピリオド（'.'）とすると，文字列"RTLU8: Run Time Library."は，"RTLU8", "Run", "Time", "Library"の 4 つのトークンに分割されます。

strtok は，*string2* に含まれる文字をデリミタとして，*string1* をいくつかのトークンに分割します。これらのルーチンを連続的に呼び出すことによって，分割されたトークンへのポインタを順番に得ることができます。*string1* に文字列へのポインタ（NULL 以外）を指定した場合，これらのルーチンは，もし *string1* の先頭に *string2* で指定されるデリミタが存在すれば，それを読み飛ばし，最初に現れるトークンへのポインタを返します。最初のトークンの終端には，null 文字（'¥0'）がセットされます。トークンが存在しなければ，NULL を返します。*string1* に NULL を指定した場合，これらのルーチンは，次のトークンをサーチします。もし，トークンが存在すれば，トークンへのポインタを返します。トークンの終端には，null 文字がセットされます。もし，トークンが存在しなければ，NULL を返します。

strtok は通常，次のような使い方をします。

（1） *string1* に検索対象の文字列を指定して，最初のトークンを得る。

（2） *string1* に NULL を指定して，次のトークンを得る。

（3） NULL を返すまで，（2）を繰り返す。

string2 の内容は，関数の呼び出しのたびに変更してもかまいません。これらのルーチンは，トークンを発見するとその終端に null 文字をセットします。したがって，*string1* の内容は変更されることに注意してください。

戻り値

トークンが存在すれば，トークンへのポインタを返します。トークンが存在しなければ，NULL を返します。

参照

strcspn strspn strchr strrchr strpbrk strstr

プログラム例

```
/*スペース，カンマ，セミコロン，コロンをデリミタとして，文字列をトークンに
分割する。token_stock[ ]に，トークンへのポインタを順番にセットする。*/
#include <string.h>

char string[] = "   TOKEN1,TOKEN2;  TOKEN3::TOKEN4  ";
char delimiter[] = " ,;:";

char *token_stock[20];

void main(void)
{
     char *token_ptr;
     int token_counter = 0;

/* 最初の呼び出し。最初のトークン TOKEN1 へのポインタを返す。*/
     token_ptr = strtok( string , delimiter);

     while(token_ptr != NULL)
     {
     token_stock[token_counter] = token_ptr; /*トークンへのポインタ*/
     ++token_counter;
     if(token_counter >= 20)
          break;
     /* ２回目以降の呼び出し。第一引数に NULL を指定する。
        TOKEN2 TOKEN3 TOKEN4 へのポインタを順番に返す。
        最後に NULL を返したら，終了する。 */
     token_ptr = strtok( NULL , delimiter);
     }

/* 結果は次のとおり。
     token_stock[0] :: "TOKEN1"
     token_stock[1] :: "TOKEN2"
     token_stock[2] :: "TOKEN3"
     token_stock[3] :: "TOKEN4"
     token_stock[4] :: NULL

     string[]は次のように変更されている。
     "   TOKEN1¥0TOKEN2¥0  TOKEN3¥0:TOKEN4¥0";
*/
}
```

strtol

マクロ・関数

機能

文字列を long 型の整数に変換します。

形式

#include <stdlib.h>

long strtol(const char ∗*s*, char ∗∗*endptr*, int *base*);

long strtol_n(const char __near ∗*s*, char __near ∗ __near ∗*endptr*, int *base*);

long strtol_f(const char __far ∗*s*, char __far ∗ __far ∗*endptr*, int *base*);

s 変換する文字列

endptr 走査するのを中止した文字を指すポインタ

base 基数

解説

strtol は，引数 *s* の指す文字列を long 型の整数値に変換し，その値を返します。文字列 *s* は次の形式に沿ったものでなければなりません。

[*white space*] [*sign*] [0] [{x|X}] [*digit*]

文字列の各部分の説明は以下のとおりです。

記号	意味
[*white space*]	タブまたはスペース（省略可能）
[*sign*]	符号（省略可能）
[0]	ゼロ（省略可能）
[{x|X}]	x または X（省略可能）
[*digit*]	数字列（省略可能）

strtol は，*base* が 2〜36 の範囲にあるとき文字列 *s* を *base* にしたがって変換します。すなわち，*base* が 16 の場合は文字列を 16 進数として解釈し'0'〜'9'，'a'〜'f'，'A'〜'F'までの文字を認識して数値に変換します。*base* が 0 の場合は，数字列の最初の 1, 2 文字で変換される値の基数が決定されます。決定されるのは以下のとおりです。

第 1 文字	第 2 文字	変換される値
0	1〜7	8 進数
0	x または X	16 進数
1〜9		10 進数

base が 1 のときや，負数または 36 を超える値の場合は 0 が返されます。これらのルーチンは，認識できない文字を読み込んだところで走査をやめ，*endptr* が NULL でなければ，その文字の位置を示すポインタを *endptr* にセットします。得られた値が long 型で表現できる範囲にない場合には LONG_MAX または LONG_MIN が返され，errno には ERANGE がセットされます。

戻り値

変換された文字列の値を返します。

参照

atof atoi atol strtod strtoul

プログラム例

```
#include <stdlib.h>

void main(void)
{
     long res;
     char *endp;

     res = strtol("0xabcdef", &endp, 16);
}
```

strtoul　　マクロ・関数

機能

文字列を unsigned long 型の整数に変換します。

形式

#include <stdlib.h>

unsigned long strtoul(const char ∗*s*, char ∗∗*endptr*, int *base*);

unsigned long strtoul_n(const char __near ∗*s*, char __near ∗ __near ∗*endptr*, int *base*);

unsigned long strtoul_f(const char __far ∗*s*, char __far ∗ __far ∗*endptr*, int *base*);

s	変換する文字列
endptr	走査するのを中止した文字を指すポインタ
base	基数

解説

strtoul は，引数 *s* の指す文字列を unsigned long 型の数値に変換し，その値を返します。文字列 *s* は次の形式に沿ったものでなければなりません。

[*white space*] [*sign*] [0] [{x|X}] [*digit*]

文字列の各部分の説明は以下のとおりです。

記号	意味
[*white space*]	タブまたはスペース（省略可能）
[*sign*]	符号（省略可能）
[0]	ゼロ（省略可能）
[{x|X}]	x または X（省略可能）
[*digit*]	数字列（省略可能）

strtoul は，*base* が 2～36 の範囲にあるとき文字列 *s* を *base* にしたがって変換します。すなわち，*base* が 16 の場合は文字列を 16 進数として解釈し'0'～'9'，'a'～'f'，'A'～'F'までの文字を認識して数値に変換します。*base* が 0 の場合は，数字列の最初の 1, 2 文字で変換される値の基数が決定されます。決定されるのは以下のとおりです。

第 1 文字	第 2 文字	変換される値
0	1～7	8 進数
0	x または X	16 進数
1～9		10 進数

base が 1 のときや，負数または 36 を超える値の場合は 0 が返されます。これらのルーチンは，認識できない文字を読み込んだところで走査をやめ，*endptr* が NULL でなければ，その文字の位置を示すポインタを *endptr* にセットします。得られた値が unsigned long 型で表現できる範囲にない場合には ULONG_MAX が返され，errno には ERANGE がセットされます。

戻り値

変換された文字列の値を返します。

参照

atof atoi atol strtod strtol

プログラム例

```
#include <stdlib.h>

void main(void)
{
     unsigned long res;
     char *endp;

     res = strtoul("0xabcdef", &endp, 16);
}
```

tan 関数

機能

タンジェント（正接）を計算します。

形式

#include <math.h>

double tan(double *x*);

x ラジアン単位の角度

解説

tan は，引数 *x* のタンジェントを計算します。

戻り値

tan は，引数 *x* のタンジェントを返します。

参照

acos asin atan atan2 cos sin

プログラム例

```
#include <math.h>

void main(void)
{
     double x;
     double res;

     x = 0.5;

     res = tan(x);
}
```

tanh　　関数

機能

ハイパボリックタンジェント（双曲線正接）を計算します。

形式

#include <math.h>

double tanh(double *x*);

x　　ラジアン単位の角度

解説

tanh は，引数 *x* のハイパボリックタンジェント，sinh(*x*)/cosh(*x*)を計算します。

戻り値

tanh は，引数 *x* のハイパボリックタンジェントを返します。

参照

acos asin atan atan2 cos cosh sin sinh tan

プログラム例

```
#include <math.h>

void main(void)
{
     double x;
     double res;

     x = 0.5;

     res = tanh(x);
}
```

tolower

マクロ・関数

機能

大文字を小文字に変換します。

形式

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

c　　1 バイト文字（0x00～0xff の整数）

解説

tolower は，*c* が大文字（'A'～'Z'）であれば，*c* を小文字（'a'～'z'）に変換します。*c* が大文字でない場合は，*c* は変更されません。

c に 0x00～0xff 以外の値を指定した場合の変換結果は不定です。

戻り値

c が大文字（'A'～'Z'）であれば，それに対応する小文字（'a'～'z'）を返します。それ以外であれば，*c* をそのまま返します。*c* が 0x00～0xff 以外の場合の戻り値は不定です。

参照

islower isupper toupper

プログラム例

```
     #include <ctype.h>

     char buffer1[] = "0123456789ABCDEFGabcdefg";
     char buffer2[64];

     void main(void)
     {
          int i;

          for(i = 0; buffer1[i] != '¥0'; ++i)
          {
               buffer2[i] = tolower(buffer1[i]);
          }
     /*
     buffer2[]の内容は，次のようになる。
     "0123456789abcdefgabcdefg"
     */
}
```

toupper　　マクロ・関数

機能

小文字を大文字に変換します。

形式

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

c　　1 バイト文字（0x00～0xff の整数）

解説

toupper は，*c* が小文字（'a'～'z'）であれば，*c* を大文字（'A'～'Z'）に変換します。*c* が小文字でない場合は，*c* は変更されません。

c に 0x00～0xff 以外の値を指定した場合の変換結果は不定です。

戻り値

c が小文字（'a'～'z'）であれば，それに対応する大文字（'A'～'Z'）を返します。それ以外であれば，*c* をそのまま返します。*c* が 0x00～0xff 以外の場合の戻り値は不定です。

参照

islower isupper tolower

プログラム例

```
     #include <ctype.h>

     char buffer1[] = "0123456789ABCDEFGabcdefg";
     char buffer2[64];

     void main(void)
     {
          int i;

          for(i = 0; buffer1[i] != '¥0'; ++i)
          {
               buffer2[i] = toupper(buffer1[i]);
          }
     /*
     buffer2[]の内容は，次のようになる。
     "0123456789ABCDEFGABCDEFG"
     */
}
```

va_arg va_end va_start　　マクロ

機能

可変個の引数リストの操作を行ないます。

形式

```
#include <stdarg.h>
void va_start( va_list ap, lastfix );
type va_arg( va_list ap, type );
void va_end( va_list ap );
```

ap　　引数を指すポインタ

lastfix　　呼び出された関数に渡す，最後の固定引数の名前

type　　データの型名

解説

va_arg, va_end，および va_start は，可変個の引数を持つ関数を作成するときに，可変個の引数リストに対する操作を容易に実現します。

va_start は，*ap* に可変引数リストの先頭を指すようにセットします。va_start は，最初に呼び出されなければなりません。

va_arg は，現在指している引数を *type* 型で取り出し，*ap* を次に進めます。*type* には va_arg が返す型名を指定します。*ap* は va_start で初期化した *ap* と同じものでなければなりません。

va_end は，引数リストのすべてを読み終わった後，その後の処理が正しく行なえるようにします。va_end は，最後に必ず呼び出さなければなりません。呼び出さなかった場合，その後の動作は保証されません。

戻り値

va_start，va_end は値を返しません。va_arg は，現在の *ap* が指している引数を返します。

参照

vfprintf vprintf vsprintf

プログラム例

```
     #include <stdarg.h>

     int res;

     void main(void)
     {
          res = total_fn(7, 1, 2, 3, 4, 5, 6, 7);
     }

     int total_fn(int num, ...)
     {
          va_list ap;
          int cnt = 0;
          int total = 0;

          va_start(ap, num);
          while(++cnt <= num)
               total += va_arg(ap, int);
          va_end(ap);
          return(total);
}
```

vsprintf　　関数

機能

フォーマットを指定してテキストを作り，文字列に書き込みます。

形式

```
#include <stdio.h>
int vsprintf( char *buffer, const char *format, va_list arglist );
int vsprintf_nn( char __near *buffer, const char __near *format, va_list arglist );
int vsprintf_nf( char __near *buffer, const char __far *format, va_list arglist );
int vsprintf_fn( char __far *buffer, const char __near *format, va_list arglist );
int vsprintf_ff( char __far *buffer, const char __far *format, va_list arglist );
```

buffer　　文字列を格納するバッファ

format　　フォーマット文字列

arglist　　引数リストへのポインタ

解説

vsprintf は，sprintf と同じような動作をしますが，引数リストではなく，引数リストへのポインタ *arglist* を受け取って，*format* の指すフォーマット文字列内の変換指定に応じて変換し，*buffer* の指す文字列へ書き込みます。文字列の末尾には null 文字を付加します。

変換指定などの詳細については，sprintf の解説を参照してください。

戻り値

vsprintf は，*buffer* に出力したバイト数を返します。ただし，末尾の null 文字は除きます。なんらかのエラーが発生した場合には，EOF を返します。

参照

sprintf va_arg va_end va_start

プログラム例

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

int inum;
double dnum;
char buf[50];

void main(void)
{
     inum = 127;
     dnum = 123.45;

     vsp(buf, "%d %f %s", inum, dnum, "Hello !!");
}

int vsp(char *s, char *fmt, ...)
{
     va_list ap;
     int cnt;

     va_start(ap, fmt);
     cnt = vsprintf(s, fmt, ap);
     va_end(ap);
     return(cnt);
}
```

4 標準入出力ルーチン リファレンス

4.1 標準入出力ルーチンについて

標準入出力ルーチンとは，標準入出力に対して文字の入出力処理を行うルーチンのことをいいます。

この章でストリームへのポインタ（FILE *）を引数として渡すライブラリルーチンがありますが，指定できるストリームは stdin，stdout，stderr の 3 つに限定されます。また，これらの FILE ポインタはすべて NEAR ポインタで扱います。

4.1.1 標準入出力ストリーム

標準入出力ストリームには次のファイル番号が割り当てられており，これらは main プログラムが呼ばれる前の初期化処理でオープンされます。

名称	マクロ名	ファイル番号
標準入力	stdin	0
標準出力	stdout	1
標準エラー	stderr	2

4.2 ライブラリリファレンス

ここでは，RTLU8 のランタイムライブラリルーチンのうち，標準入出力を扱うルーチンについて説明します。リファレンスの読み方については「1.9 ランタイムライブラリリファレンスの読み方」を参照してください。

fflush　　関数

機能

ストリームをフラッシュします。

形式

#include <stdio.h>

int fflush(FILE __near * *stream*);

stream　　ストリームへのポインタ

解説

fflush は，*stream* で指定したストリームと結合しているファイルが出力用にオープンされていれば，バッファの内容をストリームに書き込み，バッファをフラッシュします。

戻り値

fflush は，バッファをフラッシュできると 0 を返します。ただし，指定したストリームが読み出し専用にオープンされている場合は，なにもせずに 0 を返します。エラーがあった場合には EOF を返します。

参照

なし

プログラム例

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

static char reply[80];
static char buf[BUFSIZ];

void main(void)
{
     fprintf(stderr, "If you want to finish then
                         enter a string ¥"quit¥"¥n¥n");
     for( ; ; )
     {
          fprintf(stderr, "Enter a string : ");
          fflush(stderr);
          gets(reply);
          if(!strcmp(reply, "quit"))
               break;
     }
}
```

fgetc　　関数

機能

ストリームから文字を取得します。

形式

#include <stdio.h>

int fgetc(FILE __near * *stream*);

stream　　ストリームへのポインタ

解説

fgetc は，指定した入力ストリーム上の次の文字を返します。

戻り値

成功した場合は，fgetc は読み込んだ文字を符号拡張せずに int に変換してから返します。ファイルの終わりに達するか，エラーが検出されると，EOF を返します。

参照

fputc getc getchar ungetc

プログラム例

```
#include <stdio.h>

void main(void)
{
     int c;

     printf("Input a character : ");
     c = fgetc(stdin);
     printf("The character was : '%c' (%02x)¥n", c, c);
}
```

fgets　　関数

機能

ストリームから文字列を取得します。

形式

```
#include <stdio.h>
char *fgets( char *s, int n, FILE __near * stream );
char __near *fgets_n( char __near *s, int n, FILE __near * stream );
char __far *fgets_f( char __far *s, int n, FILE __near * stream );
```

s　　文字列を格納する領域を指すポインタ

n　　読み込む文字数

stream　　ストリームへのポインタ

解説

fgets は，*stream* から文字列を読み込んで *s* に格納します。読み込みは *n*-1 個の文字を読み込むか，または改行文字を読み込んだときに終了します。fgets は *s* の最後に改行文字を保存します。*s* の読み込まれた文字の最後には null ターミネータが付加されます。

戻り値

成功した場合は，*s* を返します。ファイルの終わりか，ファイルエラーの場合は NULL を返します。

参照

fputs gets

プログラム例

```
#include <stdio.h>

void main(void)
{
     char buf[80];

     printf("Input a string : ");
     fgets(buf, 80, stdin);
     printf("The string was : %s¥n", buf);
}
```

fprintf　　関数

機能

ストリームに書式化された出力を行ないます。

形式

#include <stdio.h>

int fprintf(FILE __near * *stream*, const char __near * *format* [, *argument*, ...]);

int fprintf_n(FILE __near * *stream*, const char __near * *format* [, *argument*, ...]);

int fprintf_f(FILE __near * *stream*, const char __far * *format* [, *argument*, ...]);

stream　　ストリームへのポインタ

format　　フォーマット文字列

argument　　変換指定に応じた引数

解説

fprintf は，一連の引数を受け入れ，*format* が指すフォーマット文字列内の変換指定とそれぞれの引数を対応させて変換し，書式化されたデータを *stream* に出力します。変換指定は引数の数だけなければなりません。

変換指定などの詳細については，第 3 章の sprintf の解説を参照してください。

重要

可変引数において，アドレスを参照する引数はフォーマット文字列のデータモデルにすべて揃えてください。

詳細については，第 3 章の sprintf の解説を参照してください。

戻り値

fprintf は，出力するバイト数を返します。エラーが発生すると，EOF を返します。

参照

fscanf printf putc sprintf

プログラム例

```
#include <stdio.h>

void main(void)
{
     fprintf(stdout, "integer : %d¥ncharacter : %c¥n", 123, 'A');
}
```

fputc　　関数

機能

ストリームへ 1 文字を出力します。

形式

#include <stdio.h>

int fputc(int c, FILE __near * *stream*);

c　　文字

stream　　ストリームへのポインタ

解説

fputc は，指定されたストリームに文字 *c* を出力します。

戻り値

成功すると，fputc は文字 *c* を返します。エラーが発生すると EOF を返します。

参照

fgetc putc

プログラム例

```
#include <stdio.h>

char s[ ] = "This is a test.¥n";

void main(void)
{
     int i;

     for(i = 0; s[i] != '¥0'; i++)
     fputc(s[i], stdout);
}
```

fputs　　関数

機能

ストリームに文字列を出力します。

形式

#include <stdio.h>

int fputs(const char * *s*, FILE __near * *stream*);

int fputs_n(const char __near * *s*, FILE __near * *stream*);

int fputs_f(const char __far * *s*, FILE __near * *stream*);

s　　文字列

stream ストリームへのポインタ

解説

fputs は，null ターミネータで終了する文字列 *s* を，指定の出力ストリームに出力します。fputs は，改行文字を追加せず，また最後の null ターミネータは出力されません。

戻り値

成功すると，fputs は負でない値を返します。失敗した場合には，EOF を返します。

参照

fgets gets puts

プログラム例

```
#include <stdio.h>

void main(void)
{
     fputs("This is a test.¥n", stdout);
}
```

fread 関数

機能

ストリームからデータを読み込みます。

形式

```
#include <stdio.h>
size_t fread( void *buffer, size_t size, size_t n, FILE __near * stream );
size_t fread_n( void __near *buffer, size_t size, size_t n, FILE __near * stream );
size_t fread_f( void __far *buffer, size_t size, size_t n, FILE __near * stream );
```

buffer データを格納する領域

size 1 項目のサイズ

n 読み込む項目の最大数

stream ストリームへのポインタ

解説

fread は，*stream* で示される入力ストリームから *size* バイトの項目を最大 *n* 個読み込んで，*buffer* に格納します。ファイルポインタは，実際に読み込まれたバイト数だけ増加します。

戻り値

fread は，読み込んだデータの個数を返します。

参照

fwrite

プログラム例

```
#include <stdio.h>

void main(void)
{
     int i, cnt;
     char buf[80];

     cnt = fread(buf, sizeof( char ), 80, stdin);
     printf("Contents of 'buf' : ");
     for(i = 0; i < cnt; ++i)
     {
          printf("%02x ", buf[i]);
     }
}
```

fscanf　　関数

機能

入力ストリームから入力をスキャンし，書式化します。

形式

```
#include <stdio.h>
int fscanf( FILE __near * stream, const char * format [, address, ... ] );
int fscanf_n( FILE __near * stream, const char __near * format [, address, ... ] );
int fscanf_f( FILE __near * stream, const char __far * format [, address, ... ] );
```

stream　　ストリームへのポインタ

format　　フォーマット文字列

address　　変換指定に応じた引数

解説

fscanf は，ストリームから一連の入力フィールドをスキャンして，一度に 1 文字ずつ読み込みます。次に *format* が指すフォーマット文字列内の変換指定にしたがって，各フィールドが書式化されます。最後に fscanf は，*format* の後に続く各引数が示すアドレスに書式化した入力を格納します。書式指定子とアドレスの数は，入力フィールドと同じだけなければなりません。

fscanf は，通常のフィールド終了文字（空白）に達する前に特定のフィールドのスキャンを中止することがあります。また，いくつかの理由で入力を中止することがあります。

変換指定などの詳細については，第 3 章の sscanf の解説を参照してください。

重要

可変引数において，アドレスを参照する引数はフォーマット文字列のデータモデルにすべて揃えてください。

詳細については，第 3 章の sscanf の解説を参照してください。

戻り値

fscanf は，正しくスキャンし，変換し，格納した入力フィールドの数を返します。戻り値には，値を格納しなかったフィールドは含まれません。

参照

fprintf scanf sscanf

プログラム例

```
#include <stdio.h>

void main(void)
{
     int i;

     printf("Input an integer : ");
     if(fscanf(stdin, "%d", &i))
          printf("The integer : %d¥n", i);
     else
          printf("Cannot read an integer¥n");
}
```

fwrite　　関数

機能

データをストリームに書き込みます。

形式

```
#include   <stdio.h>
size_t  fwrite( const void ∗buffer, size_t size, size_t n, FILE __near ∗ stream );
size_t  fwrite_n( const void __near ∗buffer, size_t size, size_t n, FILE __near ∗ stream );
size_t  fwrite_f( const void __far ∗buffer, size_t size, size_t n, FILE __near ∗ stream );
```

buffer	書き込むデータへのポインタ
size	1 項目のサイズ
n	書き込む項目の最大数
stream	ストリームへのポインタ

解説

これらの関数は，各 *size* バイトの項目を最大 *n* 個，*buffer* で示される領域から *stream* に書き込みます。*stream* に結び付けられているファイルポインタは，実際に書き込まれたバイト数だけ増加します。

戻り値

実際に書き込んだデータの項目数を返します。エラーが発生した場合は，*n* で指定した値よりも小さくなることがあります。

参照

fread

プログラム例

```
#include <stdio.h>

void main(void)
{
     int cnt;
     char buf[80];

     cnt = fread(buf, sizeof(char), 80, stdin);
     fwrite(buf, sizeof(char), cnt, stdout);
}
```

getc　　マクロ・関数

機能

ストリームから１文字を取得します。

形式

#include <stdio.h>

int getc(FILE __near * *stream*);

stream　　ストリームへのポインタ

解説

getc は，指定の入力ストリームから次の１文字を読み込んで，そのストリームのファイルポインタを次の文字を指すようにインクリメントします。

戻り値

成功すると，getc は読み込んだ文字を符号拡張せずに int に変換してから返します。ファイルエンドまたはエラーの場合は EOF を返します。

参照

fgetc getchar gets putc putchar ungetc

プログラム例

```
#include <stdio.h>

void main(void)
{
     int c;

     printf("Input a character : ");
     c = getc(stdin);
     printf("The character was : '%c' (%02x)¥n", c, c);
}
```

getchar　　マクロ・関数

機能

標準入力（stdin）から 1 文字を取得します。

形式

#include <stdio.h>

int getchar(void);

解説

getchar は，入力ストリーム（stdin）上の次の 1 文字を返します。getchar は，getc（stdin）と同値です。

戻り値

成功すると，getchar は読み込んだ文字を符号拡張せずに int に変換してから返します。ファイルエンドまたはエラーの場合は EOF を返します。

参照

fgetc getc gets putc putchar scanf ungetc

プログラム例

```
#include <stdio.h>

void main(void)
{
     int c;

     printf("Input a character : ");
     c = getchar( );
     printf("The character was : '%c' (%02x)¥n", c, c);
}
```

gets　　関数

機能

標準入力（stdin）から文字列を読み込みます。

形式

#include <stdio.h>

char ＊gets(char ＊ *s*);

char __near ＊gets(char __near ＊ *s*);

char __far ＊gets(char __far ＊ *s*);

s　　文字列を格納する領域を指すポインタ

解説

gets は，標準入力ストリーム（stdin）から，改行文字で終了する文字列を読み込んで *s* に格納します。改行文字は，*s* 内では null 文字に置き換えられます。

gets では，入力文字列にホワイトスペース（空白，タブ）があってもかまいません。gets は，改行文字に出会うと読み込みをやめて，それまでに読み込んだすべての文字を *s* にコピーします。

戻り値

gets は，成功すると *s* を返します。エラーの場合は NULL を返します。

参照

fgets fputs getc puts scanf

プログラム例

```
#include <stdio.h>

void main(void)
{
     char buf[80];

     printf("Input a string : ");
     gets(buf);
     printf("The string was : %s¥n", buf);
}
```

printf　　関数

機能

書式化された出力を標準出力に書き出します。

形式

```
#include <stdio.h>
int printf( const char * format [, argument, ...] );
int printf_n( const char __near * format [, argument, ...] );
int printf_f( const char __far * format [, argument, ...] );
```

format　　フォーマット文字列

argument　　変換指定に応じた引数

解説

printf は *format* が指すフォーマット文字列内の変換指定をそれぞれの引数に適用して，書式化されたデータを標準出力に出力します。変換指定は引数と同じ数だけなければなりません。

変換指定などの詳細については，第 3 章の sprintf の解説を参照して下さい。

重要

可変引数において，アドレスを参照する引数はフォーマット文字列のデータモデルにすべて揃えてください。

詳細については，第 3 章の sprintf の解説を参照してください。

戻り値

printf は，出力バイト数を返します。エラーが発生すると，printf は EOF を返します。

参照

fprintf fscanf putc puts scanf sprintf vprintf vsprintf

プログラム例

```
#include <stdio.h>

void main(void)
{
     printf("integer : %d¥n"
     "floating point : %f¥n"
     "character : %c¥n", 1234, 3.14, 'A');
}
```

putc　　マクロ・関数

機能

ストリームに 1 文字を出力します。

形式

#include <stdio.h>

int putc(int *c*, FILE __near * *stream*);

c　　文字

stream　　ストリームへのポインタ

解説

putc は *stream* が指定するストリームに文字 *c* を出力します。

戻り値

putc は，成功すると出力された文字 *c* を返します。エラーが発生すると putc は EOF を返します。

参照

fprintf fputc fputs getc getchar printf putchar

プログラム例

```
#include <stdio.h>

char s[ ] = "This is a test.¥n";

void main(void)
{
     char *p = s;

     while(*p != '¥0')
     putc(*p++, stdout);
}
```

putchar　　マクロ・関数

機能

文字を標準出力（stdout）に出力します。

形式

#include <stdio.h>

int putchar(int *c*);

c　　文字

解説

putchar は，文字 *c* を標準出力に出力します。putchar（ c ）は，putc（ c, stdout ）と同値です。

戻り値

putchar は，成功すると文字 *c* を返します。エラーが発生すると，putchar は EOF を返します。

参照

getc getchar printf putc puts

プログラム例

```
#include <stdio.h>

const char s[ ] = "This is a test.¥n";

void main(void)
{
     const char *p = s;

     while(*p != '¥0')
     putchar(*p++);
}
```

puts　　関数

機能

標準出力（stdout）に文字列を出力します。

形式

#include <stdio.h>

int puts(const char * *s*);

int puts_n(const char __near * *s*);

int puts_f(const char __far * *s*);

s　　文字列

解説

puts は，null ターミネータで終わる文字列 *s* を標準出力ストリーム（stdout）に出力し，最後に改行文字を出力します。

戻り値

成功すると，puts は負でない値を返します。エラーの場合は，EOF を返します。

参照

fputs gets printf putchar

プログラム例

```
#include <stdio.h>

void main(void)
{
     puts("This is a test.");
}
```

scanf 関数

機能

標準入力ストリームをスキャンして書式付きで入力します。

形式

#include <stdio.h>

int scanf(const char * *format* [, *address* , ...]);

int scanf_n(const char __near * *format* [, *address* , ...]);

int scanf_f(const char __far * *format* [, *address* , ...]);

format フォーマット文字列

address 変換指定に応じた引数

解説

scanf は，ストリームから一連の入力フィールドをスキャンして，標準入力ストリーム（stdin）から一度に 1 文字ずつ読み込みます。次に *format* が指すフォーマット文字列内の変換指定にしたがって，各フィールドが書式化されます。最後に scanf は，*format* の後に続く各引数が示すアドレスに，書式化した入力を格納します。変換指定子とアドレスの数は，入力フィールドと同じだけなければなりません。

変換指定などの詳細については，第 3 章の sscanf の解説を参照して下さい。

重要

可変引数において，アドレスを参照する引数はフォーマット文字列のデータモデルにすべて揃えてください。

詳細については，第 3 章の sscanf の解説を参照してください。

戻り値

scanf は，正しくスキャンし，変換し，格納した入力フィールドの数を返します。戻り値には，値を格納しなかったフィールドは含まれません。

scanf がファイルエンドを読み込むと，戻り値は EOF になります。フィールドが 1 個も格納されなければ，戻り値は 0 になります。

参照

fscanf getc printf sscanf

プログラム例

```
#include <stdio.h>

void main(void)
{
     int i;

     printf("Input an interger : ");
     if(scanf("%d", &i))
          printf("The integer : %d¥n", i);
     else
          printf("Cannot read an integer¥n");
}
```

ungetc　　関数

機能

入力ストリームに１文字をプッシュバックします。

形式

```
#include <stdio.h>
int ungetc( int c, FILE __near * stream );
```

c　　文字

stream　　ストリームへのポインタ

解説

ungetc は，文字 *c* を元の指定された入力ストリーム *stream* に返し（プッシュバック）ます。この *stream* は読み込み用にオープンされていなければなりません。この文字は次にその *stream* を getc または fread で呼び出すと返されます。どんな状況でも，１文字をプッシュバックできます。getc を呼び出さずに，２度目に ungetc を呼び出すと，以前にプッシュバックした文字は消えてしまいます。

戻り値

成功すると，ungetc はプッシュバックした文字コードを返します。操作に失敗すると EOF を返します。

参照

getc

プログラム例

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

void main(void)
{
     int i = 0;
     int c;

     printf("Input an integer : ");
     while((c = getchar()) != '¥n' && isdigit(c))
          i = 10 * i + c - '0';
     ungetc(c, stdin);
     printf("i : %d, push back character : %c¥n", i, getchar());
}
```

vfprintf　　関数

機能

ストリームに書式付き出力を書き込みます。

形式

```
#include <stdio.h>
int vfprintf( FILE __near * stream, const char * format, va_list arglist );
int vfprintf_n( FILE __near * stream, const char __near * format, va_list arglist );
int vfprintf_f( FILE __near * stream, const char __far * format, va_list arglist );
```

stream　　ストリームへのポインタ

format　　フォーマット文字列

arglist　　引数リストへのポインタ

解説

これらの関数は，fprintf と同じような動作をしますが，引数リストではなく，引数リストへのポインタを受け入れます。

vfprintf は，引数のならびを指すポインタを受け取り，*format* が指すフォーマット文字列内の変換指定を各引数に適用して，書式化されたデータをストリームに出力します。変換指定の数は，引数と同じだけなければなりません。

変換指定などの詳細については，第 3 章の sprintf の解説を参照してください。

重要

可変引数において，アドレスを参照する引数はフォーマット文字列のデータモデルにすべて揃えてください。

詳細については，第 3 章の sprintf の解説を参照してください。

戻り値

vfprintf は出力したバイト数を返します。エラーが発生すると，EOF を返します。

参照

fprintf va_arg va_end va_start vprintf vsprintf

プログラム例

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

int vfprn(char * fmt, ...)
{
     va_list ap;
     int cnt;

     va_start(ap, fmt);
     cnt = vfprintf(stdout, fmt, ap);
     va_end(ap);
}

void main(void)
{
     vfprn("integer : %d¥n"
     "floating point : %f¥n"
     "character : %c¥n", 1234, 3.14, 'A');
}
```

vprintf　　関数

機能

書式付き出力を書き込みます。

形式

```
#include <stdio.h>
int vprintf( const char * format, va_list arglist );
int vprintf_n( const char __near * format, va_list arglist );
int vprintf_f( const char __far * format, va_list arglist );
```

format　　フォーマット文字列

arglist　　引数リストへのポインタ

解説

これらの関数は，printf と同じような動作をしますが，引数リストではなく，引数リストへのポインタを受け入れます。

vprintf は，引数のならびを指すポインタを受け取り，*format* が指すフォーマット文字列内の変換指定を各引数に適用して，書式化されたデータを標準出力ストリームに出力します。変換指定の数は，引数と同じだけなければなりません。

変換指定などの詳細については，第 3 章の sprintf の解説を参照してください。

重要

可変引数において，アドレスを参照する引数はフォーマット文字列のデータモデルにすべて揃えてください。

詳細については，第 3 章の sprintf の解説を参照してください。

戻り値

vprintf は出力したバイト数を返します。エラーが発生すると，EOF を返します。

参照

printf va_arg va_end va_start vfprintf vsprintf

プログラム例

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

int vprn(const char * fmt, ...)
{
     va_list ap;
     int cnt;

     va_start(ap, fmt);
     cnt = vprintf(fmt, ap);
     va_end(ap);
}

void main(void)
{
     vprn("integer : %d¥n"
     "floating point : %f¥n"
     "character : %c¥n", 1234, 3.14, 'A');
}
```

5 低水準関数

5.1 低水準関数とは

低水準関数とは，ハードウェアに依存する関数のことで，通常はライブラリルーチンから呼ばれます。『RTLU8 ランタイムライブラリリファレンス』の「第 4 章 標準入出力ルーチンリファレンス」に記載されている各ルーチンは低水準関数を内部でコールするため，その低水準関数をリンク時に指定する必要があります。次の標準入出力ルーチンは，低水準関数を必要とします。

標準入出力ルーチン	必要とする低水準関数
fgetc , fgets , fscanf , getc , getchar , gets , scanf	read
fflush , fprintf , fputc , fputs , fwrite , printf , putc , putchar , puts , vfprintf , vprintf	write

低水準関数は，ユーザの環境に合わせて作成する必要があります。

低水準関数を作成する場合には，「5.2 各低水準関数の仕様」にあわせるようにしてください。

5.2 各低水準関数の仕様

read　　低水準関数

機能

データを読み込みます。

形式

int read(int *handle*, unsigned char ＊*buffer*, unsigned int *len*);

handle　　オープンされたファイルを参照するハンドル

buffer　　読み込んだデータを格納する領域

len　　読み込む最大バイト数

解説

handle で指定されたファイルに対応する I/O ポートから *len* バイト分のデータを読み込み，ポインタ *buffer* で指定された領域に格納します。

戻り値

実際に読み込んだバイト数を返します。

参照

write

write　　低水準関数

機能

データを書き込みます。

形式

int write(int *handle*, unsigned char ＊*buffer*, unsigned int *len*);

handle　　オープンされたファイルを参照するハンドル

buffer　　書き込むデータ

len　　書き込む最大バイト数

解説

handle で指定されたファイルに対応する I/O ポートに，ポインタ *buffer* で指定された領域から *len* バイト分のデータを書き込みます。

戻り値

実際に書き込んだバイト数を返します。

参照

read

付録

1　データメモリ対応ルーチン一覧

以下に ANSI/ISO9899 C 標準のルーチンとそれに対応するデータメモリ対応ルーチンの形式を示します。

ルーチン名	形式

atof	double atof(const char *s);
	double atof_n(const char __near *s);
	double atof_f(const char __far *s);

atoi	int atoi(const char *s);
	int atoi_n(const char __near *s);
	int atoi_f(const char __far *s);

atol	long atol(const char *s);
	long atol_n(const char __near *s);
	long atol_f(const char __far *s);

bsearch	void *bsearch(const void *key, const void *base, size_t nelem, size_t size, int (*cmp)(const void *elem1, const void *elem2)); void __near *bsearch_nn(const void __near *key, const void __near *base, size_t nelem, size_t size, int (*cmp_nn)(const void __near *elem1, const void __near *elem2)); void __far *bsearch_nf(const void __near *key, const void __far *base, size_t nelem, size_t size, int (*cmp_nf)(const void __near *elem1, const void __far *elem2)); void __near *bsearch_fn(const void __far *key, const void __near *base, size_t nelem, size_t size, int (*cmp_fn)(const void __far *elem1, const void __near *elem2)); void __far *bsearch_ff(const void __far *key, const void __far *base, size_t nelem, size_t size, int (*cmp_ff)(const void __far * elem1, const void __far * elem2));

calloc	void *calloc(size_t nelem, size_t size);
	void __near *calloc_n(size_t nelem, size_t size);
	void __far *calloc_f(size_t nelem, size_t size);

fgets	char *fgets(char *s, int n, FILE __near *stream);
	char __near *fgets_n(char __near *s, int n, FILE __near *stream);
	char __far *fgets_f(char __far *s, int n, FILE __near *stream);

fprintf	int __noreg fprintf(FILE __near *stream, const char *format【, argument, ...】); int __noreg fprintf_n(FILE __near *stream, const char __near *format【, argument, ...】); int __noreg fprintf_f(FILE __near *stream, const char __far *format【, argument, ...】);

ルーチン名	形式

fputs	int fputs(char *s, FILE __near *stream);
	int fputs_n(char __near *s, FILE __near *stream);
	int fputs_f(char __far *s, FILE __near *stream);

fread	size_t fread(void *buffer, size_t size, size_t n, FILE __near *stream);
	size_t fread_n(void __near *buffer, size_t size, size_t n, FILE __near *stream);
	size_t fread_f(void __far *buffer, size_t size, size_t n, FILE __near *stream);

free	void free(void *ptr);
	void free_n(void __near *ptr);
	void free_f(void __far *ptr);

frexp	double frexp(double x, int *pexep);
	double frexp_n(double x, int __near *pexep);
	double frexp_f(double x, int __far *pexep);

fscanf	int __noreg fscanf(FILE __near *stream, const char *format【, address, ...】); int __noreg fscanf_n(FILE __near *stream, const char __near *format【, address, ...】); int __noreg fscanf_f(FILE __near *stream, const char __far *format【, address, ...】);
fwrite	size_t fwrite(const void ＊buffer, size_t size, size_t n, FILE __near ＊stream); size_t fwrite_n(const void __near ＊buffer, size_t size, size_t n, FILE __near ＊stream); size_t fwrite_f(const void __far ＊buffer, size_t size, size_t n, FILE __near ＊stream);

gets	char *gets(char *s);
	char __near *gets(char __near *s);
	char __far *gets(char __far *s);

longjmp	void longjmp(jmp_buf env, int value);
	void longjmp_n(jmp_buf_n env, int value);
	void longjmp_f(jmp_buf_f env, int value);

malloc	void *malloc(size_t size);
	void __near *malloc_n(size_t size);
	void __far *malloc_f(size_t size);

memchr	void *memchr(const void *region, int c, size_t count);
	void __near *memchr_n(const void __near *region, int c, size_t count);
	void __far *memchr_f(const void __far *region, int c, size_t count);

ルーチン名	形式

memcmp	int memcmp(const void *_region1_, const void *_region2_, size_t _count_);
	int memcmp_nn(const void __near *_region1_, const void __near *_region2_, size_t _count_);
	int memcmp_nf(const void __near *_region1_, const void __far *_region2_, size_t _count_);
	int memcmp_fn(const void __far *_region1_, const void __near *_region2_, size_t _count_);
	int memcmp_ff(const void __far *_region1_, const void __far *_region2_, size_t _count_);

memcpy	void *memcpy(void *_dest_, const void *_src_, size_t _count_);
	void __near *memcpy_nn(void __near *_dest_, const void __near *_src_, size_t _count_);
	void __near *memcpy_nf(void __near *_dest_, const void __far *_src_, size_t _count_);
	void __far *memcpy_fn(void __far *_dest_, const void __near *_src_, size_t _count_);
	void __far *memcpy_ff(void __far *_dest_, const void __far *_src_, size_t _count_);

memmove	void *memmove(void *_dest_, const void *_src_, size_t _count_);
	void __near *memmove_nn(void __near *_dest_, const void __near *_src_, size_t _count_);
	void __near *memmove_nf(void __near *_dest_, const void __far *_src_, size_t _count_);
	void __far *memmove_fn(void __far *_dest_, const void __near *_src_, size_t _count_);
	void __far *memmove_ff(void __far *_dest_, const void __far *_src_, size_t _count_);

memset	void *memset(void *_region_, int _c_, size_t _count_);
	void __near *memset_n(void __near *_region_, int _c_, size_t _count_);
	void __far *memset_f(void __far *_region_, int _c_, size_t _count_);

modf	double modf(double _x_, double *_pint_);
	double modf_n(double _x_, double __near *_pint_);
	double modf_f(double _x_, double __far *_pint_);

printf	int __noreg printf(const char *_format_【, _argument_, ...】); int __noreg printf_n(const char __near *_format_【, _argument_, ...】); int __noreg printf_f(const char __far *_format_【, _argument_, ...】);

puts	int puts(const char *_s_);
	int puts_n(const char __near *_s_);
	int puts_f(const char __far *_s_);

qsort	void qsort(void *_base_, size_t _n_, size_t _size_, int (*_cmp_)(const void *_elem1_, const void *_elem2_)); void qsort_n(void __near *_base_, size_t _n_, size_t _size_, int (*_cmp_nn_)(const void __near *_elem1_, const void __near *_elem2_)); void qsort_f(void __far *_base_, size_t _n_, size_t _size_, int (*_cmp_ff_)(const void __far *_elem1_, const void __far *_elem2_));

ルーチン名	形式

realloc	void *realloc(void **ptr*, size_t *size*);
	void __near *realloc_n(void __near **ptr*, size_t *size*);
	void __far *realloc_f(void __far **ptr*, size_t *size*);

scanf	int __noreg scanf(const char **format*【, *address*, ...】); int __noreg scanf_n(const char __near **format*【, *address*, ...】); int __noreg scanf_f(const char __far **format*【, *address*, ...】);

setjmp	int setjmp(jmp_buf *env*);
	int setjmp_n(jmp_buf_n *env*);
	int setjmp_f(jmp_buf_f *env*);

sprintf	int __noreg sprintf(char **buffer*, const char **format*【, *argument*, ...】); int __noreg sprintf_nn(char __near **buffer*, const char __near **format*【, *argument*, ...】); int __noreg sprintf_nf(char __near **buffer*, const char __far **format*【, *argument*, ...】); int __noreg sprintf_fn(char __far **buffer*, const char __near **format*【, *argument*, ...】); int __noreg sprintf_ff(char __far **buffer*, const char __far **format*【, *argument*, ...】);
sscanf	int __noreg sscanf(const char **string*, cont char **format*【, *address*, ...】); int __noreg sscanf_nn(const char __near **string*, const char __near **format*【, *address*, ...】); int __noreg sscanf_nf(const char __near **string*, const char __far **format*【, *address*, ...】); int __noreg sscanf_fn(const char __far **string*, const char __near **format*【, *address*, ...】); int __noreg sscanf_ff(const char __far **string*, const char __far **format*【, *address*, ...】);

strcat	char *strcat(char **string1*, const char **string2*);
	char __near *strcat_nn(char __near **string1*, const char __near **string2*);
	char __near *strcat_nf(char __near **string1*, const char __far **string2*);
	char __far *strcat_fn(char __far **string1*, const char __near **string2*);
	char __far *strcat_ff(char __far **string1*, const char __far **string2*);

strchr	char *strchr(const char **string*, int *c*);
	char __near *strchr_n(const char __near **string*, int *c*);
	char __far *strchr_f(const char __far **string*, int *c*);

ルーチン名	形式

strcmp	int strcmp(const char *_string1_, const char *_string2_);
	int strcmp_nn(const char __near *_string1_, const char __near *_string2_);
	int strcmp_nf(const char __near *_string1_, const char __far *_string2_);
	int strcmp_fn(const char __far *_string1_, const char __near *_string2_);
	int strcmp_ff(const char __far *_string1_, const char __far *_string2_);

strcpy	char *strcpy(char *_string1_, const char *_string2_);
	char __near *strcpy_nn(char __near *_string1_, const char __near *_string2_);
	char __near *strcpy_nf(char __near *_string1_, const char __far *_string2_);
	char __far *strcpy_fn(char __far *_string1_, const char __near *_string2_);
	char __far *strcpy_ff(char __far *_string1_, const char __far *_string2_);

strcspn	size_t strcspn(const char *_sting1_, const char *_string2_);
	size_t strcspn_nn(const char __near *_sting1_, const char __near *_string2_);
	size_t strcspn_nf(const char __near *_sting1_, const char __far *_string2_);
	size_t strcspn_fn(const char __far *_sting1_, const char __near *_string2_);
	size_t strcspn_ff(const char __far *_sting1_, const char __far *_string2_);

strlen	size_t strlen(const char *_string_). size_t strlen_n(const char __near *_string_). size_t strlen_f(const char __far *_string_).

strncat	char *strncat(char *_string1_, const char *_string2_, size_t _count_);
	char __near *strncat_nn(char __near *_string1_, const char __near *_string2_, size_t _count_);
	char __near *strncat_nf(char __near *_string1_, const char __far *_string2_, size_t _count_);
	char __far *strncat_fn(char __far *_string1_, const char __near *_string2_, size_t _count_);
	char __far *strncat_ff(char __far *_string1_, const char __far *_string2_, size_t _count_);

strncmp	int strncmp(const char *_string1_, const char *_string2_, size_t _count_);
	int strncmp_nn(const char __near *_string1_, const char __near *_string2_, size_t _count_);
	int strncmp_nf(const char __near *_string1_, const char __far *_string2_, size_t _count_);
	int strncmp_fn(const char __far *_string1_, const char __near *_string2_, size_t _count_);
	int strncmp_ff(const char __far *_string1_, const char __far *_string2_, size_t _count_);

strncpy	char *strncpy(char *_string1_, const char *_string2_, size_t _count_);
	char __near *strncpy_nn(char __near *_string1_, const char __near *_string2_, size_t _count_);
	char __near *strncpy_nf(char __near *_string1_, const char __far *_string2_, size_t _count_);
	char __far *strncpy_fn(char __far *_string1_, const char __near *_string2_, size_t _count_);
	char __far *strncpy_ff(char __far *_string1_, const char __far *_string2_, size_t _count_);

ルーチン名	形式

strpbrk	char *strpbrk(const char **string1*, const char **string2*);
	char __near *strpbrk_nn(const char __near **string1*, const char __near **string2*);
	char __near *strpbrk_nf(const char __near **string1*, const char __far **string2*);
	char __far *strpbrk_fn(const char __far **string1*, const char __near **string2*);
	char __far *strpbrk_ff(const char __far **string1*, const char __far **string2*);

strrchr	char *strrchr(const char **string*, int *c*);
	char __near *strrchr(const char __near **string*, int *c*);
	char __far *strrchr(const char __far **string*, int *c*);

strspn	size_t strspn(const char **string1*, const char **string2*);
	size_t strspn_nn(const char __near **string1*, const char __near **string2*);
	size_t strspn_nf(const char __near **string1*, const char __far **string2*);
	size_t strspn_fn(const char __far **string1*, const char __near **string2*);
	size_t strspn_ff(const char __far **string1*, const char __far **string2*);

strstr	char *strstr(const char **string1*, const char **string2*);
	char __near *strstr_nn(const char __near **string1*, const char __near **string2*);
	char __near *strstr_nn(const char __near **string1*, const char __far **string2*);
	char __far *strstr_nn(const char __far **string1*, const char __near **string2*);
	char __far *strstr_ff(const char __far **string1*, const char __far **string2*);

strtod	double strtod(const char **s*, char ***endptr*);
	double strtod_n(const char __near **s*, char __near * __near **endptr*);
	double strtod_f(const char __far **s*, char __far * __far **endptr*);

strtok	char *strtok(char **string1*, const char **string2*);
	char __near *strtok_nn(char __near **string1*, const char __near **string2*);
	char __near *strtok_nf(char __near **string1*, const char __far **string2*);
	char __far *strtok_fn(char __far **string1*, const char __near **string2*);
	char __far *strtok_ff(char __far **string1*, const char __far **string2*);

strtol	long strtol(const char **s*, char ***endptr*, int *base*);
	long strtol_n(const char __near **s*, char __near * __near **endptr*, int *base*);
	long strtol_f(const char __far **s*, char __far * __far **endptr*, int *base*);

strtoul	unsigned long strtoul(const char **s*, char ***endptr*, int *base*);
	unsigned long strtoul_n(const char __near **s*, char __near * __near **endptr*, int *base*);
	unsigned long strtoul_f(const char __far **s*, char __far * __far **endptr*, int *base*);

ルーチン名	形式

vfprintf	int __noreg vfprintf(FILE __near **stream*, const char **format*, va_list *arglist*);
	int __noreg vfprintf_n(FILE __near **stream*, const char __near **format*, va_list *arglist*);
	int __noreg vfprintf_f(FILE __near **stream*, const char __far **format*, va_list *arglist*);

vprintf	int __noreg vprintf(const char **format*, va_list *arglist*);
	int __noreg vprintf_n(const char __near **format*, va_list *arglist*);
	int __noreg vprintf_f(const char __far **format*, va_list *arglist*);

vsprintf	int __noreg vfprintf(char **buffer*, const char **format*, va_list *arglist*);
	int __noreg vfprintf_nn(char __near **buffer*, const char __near **format*, va_list *arglist*);
	int __noreg vfprintf_nf(char __near **buffer*, const char __far **format*, va_list *arglist*);
	int __noreg vfprintf_fn(char __far **buffer*, const char __near **format*, va_list *arglist*);
	int __noreg vfprintf_ff(char __far **buffer*, const char __far **format*, va_list *arglist*);

2　処理系限界

2.1 整数値の限界

以下に，CCU8 で実装される整数値の範囲を示します。各マクロの意味ついては，「1.8.4 整数の各制限値<limits.h>」を参照してください。

定数マクロ	値
CHAR_BIT	8
SCHAR_MIN	-128
SCHAR_MAX	+127
UCHAR_MAX	255
CHAR_MIN	-128（/J オプション指定時 0）
CHAR_MAX	+127（/J オプション指定時 255）
SHRT_MIN	-32768
SHRT_MAX	+32767
USHRT_MAX	65535
INT_MIN	-32768
INT_MAX	+32767
UINT_MAX	65535
LONG_MIN	-2147483648
LONG_MAX	+2147483647
MB_LEN_MAX	2
ULONG_MAX	4294967295

2.2 浮動小数点数の限界

以下に，CCU8 で実装される浮動小数点数の特定，範囲および限界を示します。各マクロの意味ついては，「1.8.3 浮動小数点の各制限値<float.h>」を参照してください。

定数マクロ	値
DBL_DIG	15
DBL_EPSILON	2.2204460492503131e-016
DBL_MANT_DIG	53

定数マクロ	値
DBL_MAX	1.7976931348623157E+308
DBL_MAX_EXP	1024
DBL_MAX_10_EXP	308
DBL_MIN	2.2250738585072014E-308
DBL_MIN_EXP	-1021
DBL_MIN_10_EXP	-307
FLT_DIG	6
FLT_EPSILON	1.192092896E-07
FLT_MANT_DIG	24
FLT_MAX	3.402823466E+38
FLT_MAX_EXP	128
FLT_MAX_10_EXP	38
FLT_MIN	1.175494351E-38
FLT_MIN_EXP	-125
FLT_MIN_10_EXP	-37
FLT_RADIX	2
FLT_ROUNDS	1（最近値）
LDBL_DIG	15
LDBL_EPSILON	2.2204460492503131e-016
LDBL_MANT_DIG	53
LDBL_MAX	1.7976931348623157E+308
LDBL_MAX_EXP	1024
LDBL_MAX_10_EXP	308
LDBL_MIN	2.2250738585072014E-308
LDBL_MIN_EXP	-1021
LDBL_MIN_10_EXP	-307

3　CCU8 の動作

ここでは，ANSI 規格が示すところの「未定義の動作（undefined behavior ）」，「インプリメンテーション依存の動作（implementation-defined behavior ）」，および「ロケール特有の動作（locale-specific behavior ）」に対する C コンパイラ CCU8 の動作について説明します。各説明は ANSI 規格「ANSI/ISO 9899-1990 」の節に対応付けて行います。ANSI 規格では、「未定義の動作」，「インプリメンテーション依存の動作」および「ロケール特有の動作」を次のように定義しています。

未定義の動作

移植性を考えていないプログラムやエラーのあるプログラム構造，エラーのあるデータ，または不定値を含んだオブジェクトなどを使用した場合，ANSI 規格の必要条件として課されていない動作。

インプリメンテーション依存の動作

正しいプログラム構成と正しいデータに対する動作で、そのインプリメンテーションの特性に依存する動作（各種コンパイラごとに規定される動作）。

ロケール特有の動作

国籍、文化、言語といった地域の習慣に依存した動作。

3.1 未規定の動作

ANSI 規格 7.1.4 エラー<error.h>

errno はマクロでなく，外部識別子で扱います。

ANSI 規格 7.6.1.1 setjmp マクロ

setjmp は外部識別子でなく，マクロで扱います。

ANSI 規格 7.8.1.3 va_end マクロ

va_end は外部識別子でなく，マクロで扱います。

ANSI 規格 7.9.7.11 ungetc 関数

ungetc 関数には，テキストストリームは存在しませんので，そのファイル位置表示子も存在しません。

ANSI 規格 7.9.9.1 fgetpos 関数

fgetpos 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.9.9.4 ftell 関数

ftell 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.10.3 記憶域管理関数

calloc,malloc 及び realloc 関数の呼び出しによって割り付けられる記憶域の順序は下位アドレスから順に取られ，隣接性はあります。

ANSI 規格 7.10.5.1 bsearch 関数

bsearch 関数は，比較して等しい二つに要素があった場合，バイナリサーチにより最初に探索した要素へのポインタを返します。

ANSI 規格 7.10.5.2 qsort 関数

qsort 関数は，同じ要素のものをクイックソートすると，その順序が変わることもありますが、2 つの要素は連続して並びます。

ANSI 規格 7.12.2.4 time 関数

time 関数は RTLU8 ではサポートしていません。

3.2 未定義の動作

ANSI 規格 7. ライブラリ

memmove 以外のライブラリ関数の使用によって，重複オブジェクトにオブジェクトをコピーしようとした場合，重複部分のデータは保証されません。

ANSI 規格 7.1.2 標準ヘッダ

C 標準ライブラリで提供される，関数宣言，オブジェクト宣言，型定義，マクロ定義は，参照よりも前に対応するヘッダファイルをインクルードする必要があります。参照より後にインクルードした場合，リンク時にエラーが発生します。また，キーワードと同じマクロを定義した後でヘッダファイルをインクルードした場合，動作は保証されません。

ANSI 規格 7.1.3 予約済み識別子

予約済みの外部識別子を再定義している場合，コンパイラはエラーを出力します。

ANSI 規格 7.1.4 エラー<error.h>

errno は，RTLU8 では外部識別子で扱われますので，マクロ定義を無効にした場合でも errno へのアクセスは可能です。

ANSI 規格 7.1.6 共通の定義<stddef.h>

offsetof のマクロの第 2 パラメータに構造体のビットフィールドメンバを指定すると，コンパイラはエラーを出力します。

ANSI 規格 7.1.7 ライブラリ関数の使用法

ライブラリ関数の実引数が正しくない値を持っている場合，プログラムの動作は保証されま

せん。

可変個引数を受け入れる関数がインクルードされるヘッダに宣言されていない場合，正しく動作しないことがあります。この場合，コンパイラは何も出力しませんが，ワーニングレベルを 3 にしたときにワーニングを出力します。

ANSI 規格 7.2 診断機能<assert.h>

assert.h は RTLU8 ではサポートしていません。

ANSI 規格 7.3 文字操作<ctype.h>

文字列操作関数への引数が unsigned char または EOF 以外の場合，動作は保証されません。

ANSI 規格 7.6 非局所分岐<setjmp.h>

RTLU8 では、setjmp はマクロとして扱っていますので，実際の関数へのアクセスはできません。マクロ定義を無効にした場合，エラーを出力します。

ANSI 規格 7.6.1.1 setjmp マクロ

setjmp マクロは以下に示す用途で利用が推奨されます。これら以外で利用してもエラーとはなりませんが，複雑な式の中で使用した場合，現在の実行環境の一部が失われる可能性があります。

・選択文，繰り返し文，および整数定数式の比較における，オペランドの制御（単項演算子!による暗黙処理など）

・選択文や繰り返し文のオペランドの制御

・式文（void へのキャスト）

ANSI 規格 7.6.2.1 longjmp 関数

setjmp 実行から longjmp 呼び出しまでの間に，volatile 指定されていない自動記憶クラスのオブジェクトが変更された場合，そのオブジェクトの値は保証されません。

ANSI 規格 7.7.1.1 signal 関数

signal 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.8 可変個数実引数<stdarg.h>

ある関数（関数 A とする）において，va_arg マクロの引数 ap（可変個引数列）を実引数として呼び出された関数（関数 B とする）の中で，その ap を使って va_arg マクロを呼び出した場合，次のようになります。

・関数 B（ある関数 A から呼び出された側）では，呼び出された時点で ap が指す可変個引数から参照できます。

・関数 A（関数 B を呼び出した側）では，関数 B が可変個引数を参照する/しないに関わらず，関数 B を呼び出した時点の ap が指す可変個引数から参照できます。

ANSI 規格 7.8.1 可変個数実引数並びアクセスマクロ

va_start, va_arg, va_end はマクロのみで構成されており，実関数は存在しません。

ANSI 規格 7.8.1.1 va_start マクロ

va_start マクロの仮引数最終引数が，レジスタ記憶域クラス，関数型，配列型であるとき，または規定の実引数拡張を適用した型と適合しない場合，動作は保証されません。

ANSI 規格 7.8.1.2 va_arg マクロ

va_arg を呼び出したとき，処理指定の引数が実際には存在しない場合，処理指定の引数が指定された型でない場合，動作は保証されません。ap の指す領域から、引数の型のサイズだけ何らかの値を取り出します。

ANSI 規格 7.8.1.3 va_end マクロ

va_start マクロを呼び出す前に va_end マクロを呼び出しても正常に動作します。また va_end マクロを呼び出す前に，va_start マクロによって初期化された可変引数リストをもつ関数が return をした場合でも，正常に動作します。va_end はマクロ展開され，0 に置き換えられます。

ANSI 規格 7.9.5.2 fflush 関数

入力ストリームを fflush 関数に渡した場合は、fflush 関数は何もせずに 0 を返します。

ANSI 規格 7.9.5.3 fopen 関数

fopen 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.9.6 書式付き入出力関数

printf 系又は scanf 系関数の書式が実引数並びと一致しない場合，エラーとはなりませんが，動作は保証されません。書式の中に無効な変換指定がある場合，エラーとはなりませんが，不定です。

また printf 系関数において，%%の間に数字が含まれる場合のみフィールド幅として扱われます。その他のものが間に含まれる場合は不正な値を出力するか，無視されます。scanf 系関数においては、%%の間に character が含まれた場合は，動作は保証されません。

ANSI 規格 7.9.6.1 printf 系関数

printf 系関数の変換指定が d,i,n,o,u,x,X 以外の変換指定子に対して h もしくは l を含む場合，または e,E,f,g,G 以外の変換指定子に対して L を含む場合，その修飾文字は無視されます。

printf 系関数の変換指定が o,x,X,e,E,f,g,G 以外の変換指定子に対して#を含む場合，または d,i,o,u,x,X,e,E,f,g,G 以外の変換指定子に対して 0 を含む場合，そのフラグは無視されます。

集合体，共用体，またはそれらへのポインタが，printf 系の%p および%s 以外に指定された場合でも正常に動作します。

printf 系関数による単一の%変換の変換結果が 509 文字を超える文字出力となる場合，メモリの空き容量に依存するので，動作の保証はされません。

ANSI 規格 7.9.6.2 fscanf 関数

scanf 系関数の変換指定が d,i,n,o,u,x 以外の変換指定子に対して h もしくは l を含む場合，または e,f,g 以外の変換指定子に対して L を含む場合，その修飾文字は無視されます。

printf 系の%p 変換の出力形式と scanf 系の%p に代入されるアドレス形式には，書式文字列の置かれているメモリ領域が等しい場合に限り，互換性があります。

scanf 系関数による変換結果を代入する領域が，容量不足あるいは適合しない型である場合，動作は保証されません。

ANSI 規格 7.10.1 文字列変換関数

文字列を数値に変換する関数（atof, atoi, atol）の変換結果が表現できない場合，動作は保証されません。

ANSI 規格 7.10.3 記憶域管理関数

free 関数または realloc 関数の呼び出しによって解放された領域を参照した場合，動作は保証されません。

free 関数または realloc 関数のポインタ引数が calloc 関数，malloc 関数，realloc 関数の呼び出しによって返されたポインタと一致しない場合，またはそれが指す領域が free もしくは realloc の呼び出しによってすでに解放されている場合，動作は保証されません。

ANSI 規格 7.10.4.3 exit 関数

exit 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.10.6 整理算術関数

整数算術関数（abs,div,labs,ldiv）の結果が表現できない場合，動作は保証されません。

ANSI 規格 7.10.7 多バイト文字関数

これらの関数は RTLU8 ではサポートしていません。

ANSI 規格 7.11.2 複写関数　7.11.3 連結関数

複写関数または連結関数が書き込む配列のサイズが小さいと、配列の範囲を越えてコピーしてしまい、メモリの値を壊してしまうので，動作は保証されません。

ANSI 規格 7.12.3.5 strftime 関数

strftime 関数は RTLU8 ではサポートしていません。

3.3 処理系定義の動作

3.3.1 配列とポインタ

ANSI 規格 7.1.1 用語の定義

sizeof 演算子の型 size_t は，unsigned int として定義されています。

2 つの要素へのポインタ間の差を保持するために必要な整数の型 ptrdiff_t は near/far ポインタの場合，unsigned int として扱われ，huge ポインタの場合，unsigned long int として扱われます

3.3.2 ライブラリ関数

ANSI 規格 7.1.6 共通の定義<stddef.h>

マクロ NULL の定義は 0 です。

ANSI 規格 7.2 診断機能<assert.h>

assert 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.3.1 文字判定関数

isalnum, isalpha,iscntrl,islower,isupper,isprint が真を返す文字（ASCII 文字）は次の通りです。

関数名	文字セット
isalnum	0～9, A～Z, a～z
isalpha	A～Z, a～z
iscntrl	0x00～0x1F, 0x7F
islower	a～z
isupper	A～Z
isprint	0x20～0x7E

ANSI 規格 7.5.1 エラー状態の扱い

数学関数に定義域エラーが発生した場合，それぞれの数学関数は-NaN を返します。また数学関数においてアンダーフローエラーが起きた場合，errno に ERANGE はセットしません。

ANSI 規格 7.5.6.4 fmod 関数

fmod の第 2 引数が 0 の場合，ドメインエラーとなり，errno に EDOM をセットします。

ANSI 規格 7.7.1.1 signal 関数

signal 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.9.4.1 remove 関数

remove 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.9.4.2 rename 関数

rename 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.9.6.1 printf 系関数

printf 系の変換指定%p の出力は，%x の出力と等しくなります。書式文字列が FAR データの場合，物理セグメントの情報が付加されます。

ANSI 規格 7.9.6.2 scanf 系関数

scanf 系の変換%p の入力は，書式文字列が NEAR データの場合，%x の入力と等しくなります。書式文字列が FAR データの場合，%lx の入力と等しくなります。

scanf 関数中の%[変換において文字-（ハイフン）が走査文字の並び中の最初の文字でも最後の文字でもない場合，文字-は文字として扱われます。範囲指定では用いることはできません。

ANSI 規格 7.9.9.1 fgetpos 関数　7.9.9.4 ftell 関数

fgetpos 関数，ftell 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.9.10.4 perror 関数

perror 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.10.3 記憶域管理関数

calloc 関数，malloc 関数，realloc 関数に 0 バイトのサイズのメモリが要求された場合，NULL を返します。realloc 関数は，ポインタ引数が NULL でなければ，再割り当ての対象となるメモリを解放します。

ANSI 規格 7.10.4.1 abort 関数

abort 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.10.4.3 exit 関数

exit 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.10.4.4 getenv 関数

getenv 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.10.4.5 system 関数

system 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.11.6.2 strerror 関数

strerror 関数は RTLU8 ではサポートしていません。

ANSI 規格 7.12.1 時間の要素

time.h に含まれる関数は RTLU8 ではサポートしていません。

ANSI 規格 7.12.2.1 clock 関数

clock 関数は RTLU8 ではサポートしていません。

3.4 ロケール特有の動作

ANSI 規格 7.1.1 用語の定義

小数点を表す文字は，0x20('.')です。

ANSI 規格 7.3 文字操作<ctype.h>

文字操作関数の文字判定関数における，実装に依存する判定文字セットについては，「2.2.2 ライブラリ関数」にある「ANSI 規格 7.3.1 文字判定関数」の部分を参照してください。

ANSI 規格 7.11.4.4 strncmp 関数

strncmp 関数における文字セットの照合順序は ASCII と同じ順序です。文字列の先頭から比較します。

ANSI 規格 7.12.3.5 strftime 関数

time.h に含まれる関数は RTLU8 ではサポートしていません。

3.5 共通の拡張

3.5.1 付加的なストリームの型

ANSI 規格 7.9.2 ストリーム

標準入出力ストリームには次のファイル番号は割り当てられています。

名称	マクロ名	ファイル番号
標準入力	stdin	0
標準出力	stdout	1
標準エラー	stderr	2

stdin はシリアルポートの受信ポートに，stdout,stderr はシリアルポートの送信ポートに割り付けられています。

ANSI 規格 7.9.5.3 fopen 関数

fopen 関数は RTLU8 ではサポートしていません。

3.5.2 定義されたファイル位置指示子

ANSI 規格 7.9.7.11 ungetc 関数

ungetc 関数には，ファイル位置表示子は存在しません。

4　ライブラリ消費スタック一覧

以下に ANSI/ISO9899 C 標準のルーチンとそれに対応するデータメモリ対応ルーチンのスタック消費量を示します。

4.1 ROM WINDOW機能使用時

ヘッダ名	関数名	消費スタックサイズ	
		SMALL メモリモデル	LARGE メモリモデル
ctype	isalnum	2	2
	isalpha	2	2
	iscntrl	2	2
	isdigit	2	2
	isgraph	2	2
	islower	2	2
	isprint	2	2
	ispunct	2	2
	isspace	2	2
	isupper	2	2
	isxdigit	2	2
	tolower	0	0
	toupper	0	0
errno	errno	0	0
math	acos	316	328
	asin	316	328
	atan	204	214
	atan2	222	232
	ceil	90	96
	cos	240	250
	cosh	164	174
	exp	160	170
	fabs	80	86
	floor	90	96

ヘッダ名	関数名	消費スタックサイズ	
		SMALL メモリモデル	LARGE メモリモデル
math	fmod	114	120
	frexp_n	34	38
	frexp_f	34	38
	ldexp	56	60
	log	162	172
	log10	162	172
	modf_n	76	82
	modf_f	78	84
	pow	256	266
	sin	240	250
	sinh	198	208
	sqrt	166	174
	tan	192	200
	tanh	206	216
setjmp	setjmp_n	6	10
	setjmp_f	6	10
	longjmp_n	4	6
	longjmp_f	4	6
stdarg	va_start	0	0
	va_arg	0	0
	va_end	0	0
stdio	fflush	12	14
	fgetc	50 もしくは 16+read	58 もしくは 20+read
	fgets_n	62 もしくは 28+read	70 もしくは 32+read
	fgets_f	72 もしくは 38+read	80 もしくは 42+read
	fprintf_n	286	332
	fprintf_f	290	334
	fputc	52 もしくは 30+write	60 もしくは 36+write

ヘッダ名	関数名	消費スタックサイズ	
		SMALL メモリモデル	LARGE メモリモデル
stdio	fputs_n	60 もしくは 38+write	68 もしくは 44+write
	fputs_f	66 もしくは 44+write	74 もしくは 50+write
	fread_n	66 もしくは 32+read	74 もしくは 36+read
	fread_f	72 もしくは 38+read	80 もしくは 42+read
	fscanf_n	284	376
	fscanf_f	288	392
	fwrite_n	64 もしくは 42+write	72 もしくは 48+write
	fwrite_f	72 もしくは 50+write	80 もしくは 56+write
	getc	52 もしくは 18+read	62 もしくは 24+read
	getchar	52 もしくは 18+read	62 もしくは 24+read
	gets_n	58 もしくは 24+read	66 もしくは 28+read
	gets_f	68 もしくは 34+read	76 もしくは 38+read
	printf_n	286	332
	printf_f	290	334
	putc	54 もしくは 32+write	64 もしくは 40+write
	putchar	54 もしくは 32+write	64 もしくは 40+write
	puts_n	62 もしくは 40+write	72 もしくは 48+write
	puts_f	72 もしくは 50+write	82 もしくは 58+write
	scanf_n	284	376
	scanf_f	288	392
	sprintf_nn	288	334
	sprintf_nf	292	336
	sprintf_fn	290	338
	sprintf_ff	294	340
	sscanf_nn	284	376
	sscanf_nf	288	392
	sscanf_fn	284	376
	sscanf_ff	288	392
	ungetc	8	8

ヘッダ名	関数名	消費スタックサイズ	
		SMALL メモリモデル	LARGE メモリモデル
stdio	vfpintf_n	286	332
	vfpintf_f	290	334
	vprintf_n	286	332
	vprintf_f	290	334
	vsprintf_nn	288	334
	vsprintf_nf	292	336
	vsprintf_fn	290	338
	vsprintf_ff	294	340
stdlib	abs	2	2
	atof_n	208	220
	atof_f	220	232
	atoi_n	88	96
	atoi_f	102	110
	atol_n	88	96
	atol_f	102	110
	bsearch_nn	22 もしくは 16+_Cmpfun_nn	24 もしくは 18+_Cmpfun_nn
	bsearch_nf	32 もしくは 26+_Cmpfun_nf	38 もしくは 32+_Cmpfun_nf
	bsearch_fn	26 もしくは 20+_Cmpfun_fn	32 もしくは 26+_Cmpfun_fn
	bsearch_ff	32 もしくは 26+_Cmpfun_ff	38 もしくは 32+_Cmpfun_ff
	calloc_n	42	48
	calloc_f	74	80
	div	28	34
	free_n	8	8
	free_f	20	20
	labs	6	6
	ldiv	50	56
	malloc_n	34	38
	malloc_f	58	62

ヘッダ名	関数名	消費スタックサイズ	
		SMALL メモリモデル	LARGE メモリモデル
stdlib	qsort_n	62 もしくは 54+_Cmpfun_nn	76 もしくは 68+_Cmpfun_nn
	qsort_f	88 もしくは 76+_Cmpfun_ff	100 もしくは 88+_Cmpfun_ff
	rand	26	30
	realloc_n	52	58
	realloc_f	86	92
	srand	0	0
	strtod_n	208	220
	strtod_f	220	232
	strtol_n	82	88
	strtol_f	92	98
	strtoul_n	66	72
	strtoul_f	76	82
string	memchr_n	10	10
	memchr_f	10	10
	memcmp_nn	6	6
	memcmp_nf	10	10
	memcmp_fn	8	8
	memcmp_ff	8	8
	memcpy_nn	8	8
	memcpy_nf	12	12
	memcpy_fn	12	12
	memcpy_ff	12	12
	memmove_nn	10	10
	memmove_nf	18	18
	memmove_fn	14	14
	memmove_ff	14	14
	memset_n	12	12
	memset_f	12	12

ヘッダ名	関数名	消費スタックサイズ	
		SMALL メモリモデル	LARGE メモリモデル
string	strcat_nn	6	6
	strcat_nf	8	8
	strcat_fn	12	12
	strcat_ff	12	12
	strchr_n	6	6
	strchr_f	8	8
	strcmp_nn	4	4
	strcmp_nf	6	6
	strcmp_fn	8	8
	strcmp_ff	8	8
	strcpy_nn	6	6
	strcpy_nf	8	8
	strcpy_fn	12	12
	strcpy_ff	12	12
	strcspn_nn	8	8
	strcspn_nf	10	10
	strcspn_fn	10	10
	strcspn_ff	14	14
	strlen_n	4	4
	strlen_f	6	6
	strncat_nn	10	10
	strncat_nf	10	10
	strncat_fn	12	12
	strncat_ff	12	12
	strncmp_nn	6	6
	strncmp_nf	8	8
	strncmp_fn	8	8
	strncmp_ff	8	8

ヘッダ名	関数名	消費スタックサイズ	
		SMALL メモリモデル	LARGE メモリモデル
string	strncpy_nn	8	8
	strncpy_nf	10	10
	strncpy_fn	12	12
	strncpy_ff	12	12
	strpbrk_nn	6	6
	strpbrk_nf	8	8
	strpbrk_fn	10	10
	strpbrk_ff	12	12
	strrchr_n	8	8
	strrchr_f	10	10
	strspn_nn	10	10
	strspn_nf	10	10
	strspn_fn	12	12
	strspn_ff	14	14
	strstr_nn	16	18
	strstr_nf	20	22
	strstr_fn	24	26
	strstr_ff	28	30
	strtok_nn	22	24
	strtok_nf	26	28
	strtok_fn	24	26
	strtok_ff	28	30

RTLU8 ランタイムライブラリ
リファレンスマニュアル

2011 年 10 月　　第 4 版発行
