お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

お客様各位

資料中の「三菱電機」、「三菱XX」等名称の株式会社ルネサス テクノロジへの変更について

2003年4月1日を以って株式会社日立製作所及び三菱電機株式会社のマイコン、ロジック、アナログ、ディスクリート半導体、及びDRAMを除くメモリ(フラッシュメモリ・SRAM等)を含む半導体事業は株式会社ルネサス テクノロジに承継されました。
従いまして、本資料中には「三菱電機」、「三菱電機株式会社」、「三菱半導体」、「三菱XX」といった表記が残っておりますが、これらの表記は全て「株式会社ルネサス　テクノロジ」に変更されておりますのでご理解の程お願い致します。尚、会社商標・ロゴ・コーポレートステートメント以外の内容については一切変更しておりませんので資料としての内容更新ではありません。

注:「高周波・光素子事業、パワーデバイス事業については三菱電機にて引き続き事業運営を行います。」

2003年4月1日
株式会社ルネサス テクノロジ
カスタマサポート部

User's Manual

TW32R V.3.10
標準ライブラリマニュアル

M32R ファミリ用　クロスツールキット(GNU 版)

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2000.06

《安全設計に関するお願い》

三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

《本資料ご利用に際しての留意事項》

- 本資料は、お客様が用途に応じた適切な三菱半導体製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について三菱電機株式会社・三菱電機セミコンダクタシステム株式会社が所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は責任を負いません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は、予告なしに、本資料に記載した製品または仕様を変更することがあります。三菱半導体製品のご購入に当たりましては、事前に三菱電機株式会社・三菱電機セミコンダクタシステム株式会社または特約店へ最新の情報をご確認頂きますとともに、三菱電機半導体情報ホームページ（http://www.semicon.melco.co.jp/）および三菱開発ツールホームページ（http://www.tool-spt.mesc.co.jp/）などを通じて公開される情報に常にご注意ください。
- 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社はその責任を負いません。
- 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は、適用可否に対する責任は負いません。
- 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社または特約店へご照会ください。
- 本資料の転載、複製については、文書による三菱電機株式会社・三菱電機セミコンダクタシステム株式会社の事前の承諾が必要です。
- 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたら三菱電機株式会社・三菱電機セミコンダクタシステム株式会社または特約店までご照会ください。

製品の内容及び本書についてのお問い合わせ先

電子メールの場合：インストーラが生成する以下のテキストファイルに必要事項を記入の上、開発ツールサポート窓口support@tool.mesc.co.jpまで送信ください。

Windows 98/95/Windows NT 4.0版：\SUPPORT\製品名\SUPPORT.TXT
EWS版：/support/製品名/toolinfo.txt

FAXの場合：各ユーザーズマニュアル及びガイドブックの最後に添付されている「技術サポート連絡書」に必要事項を記入の上、開発ツールサポート窓口まで送信ください。FAX送信先は「技術サポート連絡書」に記載してあります。

Short Contents

Table of Contents

はじめに

このマニュアルをお読みになる前に、製品に添付されているリリースノートを必ずお読みください。製品構成、製品の扱い、注意事項などの重要な内容が記述されています。

■対象読者

このマニュアルでは、次の方を対象としています。

- C 言語によるプログラム開発経験のある方
- 三菱製 ANSI-C 標準ライブラリをご使用になる方

1 三菱 ANSI-C 標準ライブラリ概要

1.1 C 標準ライブラリの種類

三菱製 ANSI-C 標準ライブラリは、次の 2 つのライブラリから構成されています。

'`libc.a`' 標準ライブラリ

'`libm.a`' 数値計算ライブラリ

これらのライブラリは、M32R コアと M32Rx コアの各メモリモデル毎 (large モデルを除く) にライブラリが提供されています。TW32R をデフォルトのインストール先 (*c:\mtool*) にインストールした場合、それぞれ下の表のディレクトリ[1] にインストールされます。

■ M32R 用

small 用 *c:\mtool*\tw32r\h-i386-cygwin32\m32r-elf\lib\libc.a
c:\mtool\tw32r\h-i386-cygwin32\m32r-elf\lib\libm.a

medium 用 *c:\mtool*\tw32r\h-i386-cygwin32\m32r-elf\lib\medium\libc.a
c:\mtool\tw32r\h-i386-cygwin32\m32r-elf\lib\medium\libm.a

large 用 デフォルトでは medium 用が使用されます。
large モデル用ライブラリは標準では提供していませんので、添付のソースファイルから構築してリンク指定する必要があります。

■ M32Rx 用

small 用 *c:\mtool*\tw32r\h-i386-cygwin32\m32r-elf\lib\m32rx\libc.a
c:\mtool\tw32r\h-i386-cygwin32\m32r-elf\lib\m32rx\libm.a

medium 用 *c:\mtool*\tw32r\h-i386-cygwin32\m32r-elf\lib\medium\m32rx\libc.a

c:\mtool\tw32r\h-i386-cygwin32\m32r-elf\lib\medium\m32rx\libm.a

large 用 デフォルトでは medium 用が使用されます。
large モデル用ライブラリは標準では提供していませんので、添付のソースファイルから構築してリンク指定する必要があります。

[1] 標準ライブラリをご使用になる場合には、リンクするライブラリのパスを間違えないように注意してください。

1.2 その他のライブラリ

1.2.1 浮動小数点演算ライブラリ (libmfp.a)

ANSI-C の標準関数以外のライブラリとして、浮動小数点演算ライブラリ libmfp.a(三菱製)を提供しています。
このライブラリは、浮動小数点型を用いたプログラムやライブラリ関数で使用されます。これらのプログラムとリンクする場合には、この libmfp.a ライブラリもリンク指定するようにしてください。

TW32R をデフォルトのインストール先 (*c:\mtool*) にインストールした場合、次のディレクトリ[2]にインストールされます。

■ M32R 用

small 用　*c:\mtool*\tw32r\h-i386-cygwin32\lib\gcc-lib\m32r-elf\2.9-gnupro-99r1p1\libmfp.a

medium 用　*c:\mtool*\tw32r\h-i386-cygwin32\lib\gcc-lib\m32r-elf\2.9-gnupro-99r1p1\medium\libmfp.a

large 用　medium 用を使用します。

■ M32Rx 用

small 用　*c:\mtool*\tw32r\h-i386-cygwin32\lib\gcc-lib\m32r-elf\2.9-gnupro-99r1p1\m32rx\libmfp.a

medium 用　*c:\mtool*\tw32r\h-i386-cygwin32\lib\gcc-lib\m32r-elf\2.9-gnupro-99r1p1\medium\m32rx\libmfp.a

large 用　medium 用を使用します。

1.2.2 GNU 組み込みライブラリ (libgcc.a)

GNU コンパイラに付属している libgcc.a ライブラリがあります。このライブラリには、GNU コンパイラが生成する組み込み関数のコードが含まれています。したがって、プログラムをリンクする場合には、この libgcc.a ライブラリもリンク指定するようにしてください。

TW32R をデフォルトのインストール先 (*c:\mtool*) にインストールした場合、次のディレクトリ[3]にインストールされます。

[2] 2.9-gnupro-99r1p1 の部分は、TW32R のバージョンによって異なる場合があります。リンク指定する際は、ご使用のバージョンにあったパスを指定するようにしてください。

[3] 2.9-gnupro-99r1p1 の部分は、TW32R のバージョンによって異なる場合があります。リンク指定する際は、ご使用のバージョンにあったパスを指定するようにしてください。現在、large モデル用は提供していません。

■ M32R 用

small 用　*c:\mtool*\tw32r\h-i386-cygwin32\lib\gcc-lib\m32r-elf\2.9-gnupro-99r1p1\libgcc.a

medium 用　*c:\mtool*\tw32r\h-i386-cygwin32\lib\gcc-lib\m32r-elf\2.9-gnupro-99r1p1\medium\libgcc.a

large 用　medium 用を使用します。

■ M32Rx 用

small 用　*c:\mtool*\tw32r\h-i386-cygwin32\lib\gcc-lib\m32r-elf\2.9-gnupro-99r1p1\m32rx\libgcc.a

medium 用　*c:\mtool*\tw32r\h-i386-cygwin32\lib\gcc-lib\m32r-elf\2.9-gnupro-99r1p1\medium\m32rx\libgcc.a

large 用　medium 用を使用します。

1.3 C 標準ライブラリ関数の分類

C 標準ライブラリは、ANSI 仕様 (ANSI/ISO 9899-1990) に準拠しています。各ライブラリに含まれる関数は、機能別に全部で 11 種類に分類されます。各種関数は、対応する標準ヘッダファイルを各処理単位ごとにインクルードすることによって使用可能となります。

■ libc.a

分類	機能概要	対応するヘッダファイル
プログラム診断関数	プログラムの診断情報の出力	assert.h
文字操作関数	文字の操作およびチェック	ctype.h
プログラムの制御移動関数	関数間の制御の移動	setjmp.h
可変個の実引数アクセス関数	可変個の実引数を持つ関数における実引数へのアクセス	stdarg.h
入出力関数	入出力操作	stdio.h
標準処理関数	メモリ管理等の C プログラムでの標準的処理	stdlib.h
文字列操作関数	文字列の比較、複写等	string.h
ロケール操作関数	ロケール (地域化) の設定、操作	locale.h

日付・時刻操作関数	日付、時刻の設定、変更等	time.h
シグナル処理関数	シグナル (割り込み) の送受信	signal.h

■ libm.a

分類	機能概要	対応するヘッダファイル
数値計算関数	三角関数等の数値計算	math.h

この他に、上記関数を使用するための諸設定を行う「非 ANSI 関数」が用意されています。非 ANSI 関数は、関数名がアンダースコア (_) で始まります。通常、スタートアッププログラム内で使用します。

1.4 C 標準ライブラリ使用時の注意

C 標準ライブラリを使用するにあたって、以下の事項に注意してください。

■ 標準ヘッダファイルをインクルードしてください。
C 標準ライブラリを使用する場合、必要な標準ヘッダファイルをインクルードしてください。

■ C 標準ライブラリ関数と同名の関数を作らないでください。
ユーザープログラム中では、C 標準ライブラリの関数と同名の関数を作らないようにしてください。ユーザーが ANSI-C 仕様と同名の関数を作ることは推奨されません。

1.5 C 標準ライブラリのエラーメッセージ

ライブラリ関数の中には、ライブラリ関数を実行中にエラーが発生した場合、標準ヘッダファイル errno.hで定義している *errno* にエラー番号を設定する関数があります。エラー番号には対応するエラーメッセージが定義してあり、そのエラーメッセージを出力することができます。*errno* をチェックするプログラム例を以下に示します。

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

void foo(void)
{
  double x,a=2.0;
```

```
    x = asin(a);
    if(errno == EDOM)
        printf("%s\n", strerror(errno));   /* print error message */
}
```

上記の例では、asin関数を使って a の逆正弦値を求めています。引数 a が asin関数の定義域 [-1, 1] を超えているとき、*errno* に値 EDOMが設定されるので、printf関数でエラーメッセージを出力しています。strerror関数は、エラー番号を実引数として渡すと、対応するエラーメッセージの文字列ポインタを返す関数です。

2 標準ヘッダファイル

2.1 標準ヘッダファイルの概要

標準ヘッダファイルとは、C 標準ライブラリ関数を仕様するために必要なプロトタイプ宣言、マクロ定義、および、データ型宣言が記述されているファイルです。C 標準ライブラリ関数を使用する場合、ライブラリ関数の実行に必要な定義や宣言を行っているヘッダファイルを、処理単位ごとにインクルードする必要があります。

以下に、標準ヘッダファイル、および、それぞれのヘッダファイルに対応するライブラリ関数 (分類名) を示します。

ヘッダファイル	概要	対応するライブラリ関数
assert.h	プログラムの診断情報の出力を行うマクロ定義	プログラム診断関数
ctype.h	文字操作関数や文字チェック関数のマクロ定義	文字操作関数
errno.h	エラー番号に関するマクロ定義	全関数 (必要に応じて)
float.h	浮動小数点数の内部表現に関する各種制限値マクロの定義	数値計算関数など (float.h のマクロ使用時のみ)
limits.h	コンパイラの内部処理に関する各種制限値マクロの定義	全関数 (必要に応じて)
locale.h	ロケール (地域化) 操作関数の宣言	ロケール (地域化) 操作関数
math.h	数値計算関数の宣言、マクロ定義	数値計算関数
setjmp.h	分岐関数の宣言、データ型宣言	プログラムの制御移動関数
signal.h	シグナル (割り込み) 処理関数の宣言	シグナル処理関数
stdarg.h	可変個の実引数を持つ関数のマクロ定義、データ型宣言	可変個の実引数アクセス関数
stddef.h	各標準ヘッダで共通に使用する定義、データ型宣言	全関数 (必要に応じて)
stdio.h	入出力関数の宣言、データ型宣言、マクロ定義	入出力関数
stdlib.h	メモリ管理等の C プログラムでの標準的処理関数の宣言、データ型宣言、マクロ定義	標準処理関数
string.h	文字列操作関数やメモリ操作関数の宣言	文字列操作関数

time.h	日付および時刻操作関数の宣言	日付・時刻操作関数

2.2 標準ヘッダファイルの詳細

ここでは、各標準ヘッダファイルにおいて、宣言または定義している標準ライブラリ関数を示します。これらの関数を使用する場合は、必ず対応するヘッダファイルのインクルードが必要です。また、制限値を定義しているマクロがある場合はマクロ名の一覧を示します。ヘッダファイルはアルファベット順に掲載してあります。

2.2.1 assert.h

プログラム診断用関数 assert を定義しています (引数を持つマクロ定義)。

2.2.2 ctype.h

文字操作関数のプロトタイプ宣言およびマクロ定義を行っています。宣言している関数は以下のとおりです。

関数	概要
isalnum	英字または 10 進数字かどうかを判定します。
isalpha	英字かどうかを判定します。
iscntrl	制御文字かどうかを判定します。
isdigit	10 進数字かどうかを判定します。
isgraph	空白を除く印字文字かどうかを判定します。
islower	英小文字かどうかを判定します。
isprint	空白を含む印字文字かどうかを判定します。
ispunct	特殊文字かどうかを判定します。
isspace	空白類文字かどうかを判定します。
isupper	英大文字かどうかを判定します。
isxdigit	16 進数字かどうかを判定します。
tolower	英大文字を英小文字に変換します。
toupper	英小文字を英大文字に変換します。

2.2.3 errno.h

エラーが発生したときに、エラー番号を保持する外部変数 *errno* と、エラー番号を表すマクロ定義を行っています。定義しているマクロ名は以下のとおりです。

マクロ名	説明
EDOM	入力引数の値が関数内で定義されている値の範囲外であるとき、*errno* に設定する値を示しています。
ERANGE	関数の計算結果が double 型の値として表せないとき、あるいはオーバーフロー/アンダフローとなったとき、*errno* に設定する値を示しています。

2.2.4 float.h

浮動小数点数の内部表現に関する各種制限値のマクロを定義しています。定義しているマクロ名および定義値は以下のとおりです。

マクロ名	説明
FLT_RADIX	指数部表現における基数を示します。
FLT_ROUNDS	加算演算結果の丸めのモードを示します。本マクロの定義の意味は以下のとおりです。 • 最も近い値（ 最近値）に丸める： 1 • +∞方向に丸める： 2 • -∞方向に丸める： 3 • 0 方向に丸める： 0 • 特に規定しない： -1 丸め、切り捨ての方法は処理系定義です。
FLT_MAX	float 型の浮動小数点数値として表現できる正の最大値を示します。
DBL_MAX	double 型の浮動小数点数値として表現できる正の最大値を示します。
LDBL_MAX	long double 型の浮動小数点数値として表現できる正の最大値を示します。
FLT_MIN	float 型の浮動小数点数値として表現できる正の値での最小値を示します。
DBL_MIN	double 型の浮動小数点数値として表現できる正の値での最小値を示します。
LDBL_MIN	long double 型の浮動小数点数値として表現できる正の値での最小値を示します。
FLT_MAX_EXP	float 型の浮動小数点数値として表現できる基数の累乗の最大値を示します。

DBL_MAX_EXP	double 型の浮動小数点数値として表現できる基数の累乗の最大値を示します。
LDBL_MAX_EXP	long double 型の浮動小数点数値として表現できる基数の累乗の最大値を示します。
FLT_MIN_EXP	float 型として表現できる浮動小数点数値の基数の累乗の最小値を示します。
DBL_MIN_EXP	double 型として表現できる浮動小数点数値の基数の累乗の最小値を示します。
LDBL_MIN_EXP	long double 型として表現できる浮動小数点数値の基数の累乗の最小値を示します。
FLT_MAX_10_EXP	float 型の浮動小数点数値として表現できる 10 の累乗の最大値を示します。
DBL_MAX_10_EXP	double 型の浮動小数点数値として表現できる 10 の累乗の最大値を示します。
LDBL_MAX_10_EXP	long double 型の浮動小数点数値として表現できる 10 の累乗の最大値を示します。
FLT_MIN_10_EXP	float 型として表現できる浮動小数点数値の 10 の累乗の最小値を示します。
DBL_MIN_10_EXP	double 型として表現できる浮動小数点数値の 10 の累乗の最小値を示します。
LDBL_MIN_10_EXP	long double 型として表現できる浮動小数点数値の 10 の累乗の最小値を示します。
FLT_DIG	float 型の浮動小数点数値の 10 進精度の最大桁数を示します。
DBL_DIG	double 型の浮動小数点数値の 10 進精度の最大桁数を示します。
LDBL_DIG	long double 型の浮動小数点数値の 10 進精度の最大桁数を示します。
FLT_MANT_DIG	float 型の浮動小数点数値を基数に合わせて表現したときの仮数部の最大桁数を示します。
DBL_MANT_DIG	double 型の浮動小数点数値を基数に合わせて表現したときの仮数部の最大桁数を示します。
LDBL_MANT_DIG	long double 型の浮動小数点数値を基数に合わせて表現したときの仮数部の最大桁数を示します。
FLT_EPSILON	float 型において、$1.0 + x \neq 1.0$ である最小の浮動小数点数値 x を示します。

DBL_EPSILON	double 型において、1.0 + x ≠ 1.0 である最小の浮動小数点数値 x を示します。
LDBL_EPSILON	long double 型において、1.0 + x ≠ 1.0 である最小の浮動小数点数値 x を示します。

2.2.5 limits.h

各型の数値上の制限値に関するマクロ定義を行っています。定義しているマクロは以下のとおりです。

マクロ名	説明
CHAR_BIT	char 型が何ビットから構成されるかを示します。
CHAR_MAX	char 型の変数が値として持つことのできる最大値を示します。
CHAR_MIN	char 型の変数が値として持つことのできる最小値を示します。
SCHAR_MAX	signed char 型の変数が値として持つことのできる最大値を示します。
SCHAR_MIN	signed char 型の変数が値として持つことのできる最小値を示します。
UCHAR_MAX	unsigned char 型の変数が値として持つことのできる最大値を示します。
SHRT_MAX	short int 型の変数が値として持つことのできる最大値を示します。
SHRT_MIN	short int 型の変数が値として持つことのできる最小値を示します。
USHRT_MAX	unsigned short int 型の変数が値として持つことのできる最大値を示します。
INT_MAX	int 型の変数が値として持つことのできる最大値を示します。
INT_MIN	int 型の変数が値として持つことのできる最小値を示します。
UINT_MAX	unsigned int 型の変数が値として持つことのできる最大値を示します。
LONG_MAX	long 型の変数が値として持つことのできる最大値を示します。
LONG_MIN	long 型の変数が値として持つことのできる最小値を示します。
ULONG_MAX	unsigned long 型の変数が値として持つことのできる最大値を示します。

2.2.6 locale.h

プログラムの地域化を操作する関数 (ロケール操作関数) のプロトタイプ宣言およびマクロ定義を行っています。宣言している関数および定義しているマクロ名は以下のとおりです。

関数	概要
localeconv	lconv 型構造体を初期化します。
setlocale	ロケール情報の設定、検索をします。

マクロ名	説明
LC_ALL	すべてのロケール情報を設定、検索します。
LC_COLLATE	strcoll 関数，strxfrm 関数に影響を与える情報を設定、検索します。
LC_CTYPE	isdigit 関数と isxdigit 関数を除くすべての文字操作関数とマルチバイトを扱う関数すべてに影響を与える情報を設定、検索します。
LC_MONETARY	localeconv 関数が返す通貨情報に影響を与える情報を設定、検索します。
LC_NUMERIC	入出力関数と文字列操作関数で使用される小数点、および localeconv 関数が返す通貨情報以外の情報に影響を与える情報を設定、検索します。
LC_TIME	strftime 関数に影響を与える情報を設定、検索します。

2.2.7 math.h

数値計算関数のプロトタイプ宣言およびマクロ定義を行っています。宣言している関数および定義しているマクロは以下のとおりです。

関数	概要
acos	浮動小数点数の逆余弦を計算します。
asin	浮動小数点数の逆正弦を計算します。
atan	浮動小数点数の逆正接を計算します。
atan2	浮動小数点数どうしを除算した結果の値の逆正接を計算します。
ceil	浮動小数点数の小数点以下を切り上げた整数値を求めます。

cos	浮動小数点数のラディアン値の余弦を計算します。
cosh	浮動小数点数の双曲線余弦を計算します。
exp	浮動小数点数の指数関数を計算します。
fabs	浮動小数点数の絶対値を計算します。
floor	浮動小数点数の小数点以下を切り捨てた整数値を求めます。
fmod	浮動小数点数どうしを除算した結果の余りを計算します。
frexp	浮動小数点数を(0.5, 1.0）の値と 2 の累乗の積に分解します。
ldexp	浮動小数点数と 2 の累乗の乗算を計算します。
log	浮動小数点数の自然対数を計算します。
log10	浮動小数点数の 10 を底をする対数を計算します。
modf	浮動小数点数を整数部分と小数部分に分解します。
pow	浮動小数点数の累乗を計算します。
sin	浮動小数点数のラディアン値の正弦を計算します。
sinh	浮動小数点数の双曲線正弦を計算します。
sqrt	浮動小数点数の正の平方根を計算します。
tan	浮動小数点数のラディアン値の正接を計算します。
tanh	浮動小数点数の双曲線正接を計算します。

マクロ名	説明
HUGE_VAL	関数の計算結果がオーバーフローしたときに、関数のリターン値として返す値を示しています。

2.2.8 setjmp.h

分岐関数のプロトタイプ宣言および必要なデータ型宣言を行っています。宣言している関数およびデータ型は以下のとおりです。

関数	概要
longjmp	setjmp 関数で退避していた関数の実行環境を回復し、setjmp 関数を呼び出したプログラムの位置に制御を移動します。
setjmp	現在実行中の関数の実行環境を指定した記憶域に退避します。

データ型	概要
jmp_buf	関数間の制御の移動を可能とする情報を保存しておくための記憶域に対応する型名を示しています。

2.2.9 signal.h

シグナル処理関数のプロトタイプ宣言およびマクロ定義を行っています。宣言している関数は以下のとおりです。

関数	概要
signal	シグナルに応答する関数を設定します。
raise	プログラムに対してシグナルを送ります。

2.2.10 stdarg.h

可変個の実引数を持つ関数に必要なマクロ定義およびデータ型宣言を行っています。定義しているマクロおよび宣言しているデータ型は以下のとおりです。

マクロ名	説明
va_start	可変個の引数の参照を行うため、初期設定処理を行います。
va_arg	可変個の引数から順に引数を取り出します。
va_end	可変個の引数を持つ関数の引数への参照を終了します。

データ型	概要
va_list	可変個の引数を参照するために、va_start, va_arg, va_end マクロで共通に使用される変数の型を示しています。

2.2.11 stddef.h

各標準ヘッダファイルで共通に使用するマクロの定義、共通で使用するデータ型の宣言を行っています。定義しているマクロおよび宣言しているデータ型は以下のとおりです。

マクロ名	説明

errno	ライブラリ関数の処理中にエラーが発生した場合、そのライブラリごとに定義されたエラーコードがこの errno に設定されます。ライブラリ関数を呼び出す前に errno に 0 を設定しておき、ライブラリ関数の処理終了後に errno に設定されているコードを調べることによってライブラリ関数の処理中に発生したエラーをチェックすることができます。
NULL	ポインタが何も指していないときの値を示します。

データ型	概要
ptrdiff_t	二つのポインタを減算した結果の型を示します。
size_t	sizeof 演算子による演算結果の型を示します。

2.2.12 stdio.h

入出力関数のプロトタイプ宣言、および、入出力関数に必要なマクロ定義とデータ型の宣言を行っています。宣言している関数、定義しているマクロ、および、宣言しているデータ型は以下のとおりです。

関数	概要
clearerr	ストリーム入出力用ファイルのエラー状態をクリアします。
fclose	ストリーム入出力用ファイルをクローズします。
feof	ストリーム入出力用ファイルが終わりであるかどうかを判定します。
ferror	ストリーム入出力用ファイルがエラー状態であるかどうかを判定します。
fflush	ストリーム入出力用ファイルの内容をファイルへ出力します。
fgetc	ストリーム入出力用ファイルから 1 文字入力します。
fgetpos	現在のファイルストリームの位置を求めます。
fgets	ストリーム入出力用ファイルから文字列を入力します。
fopen	ストリーム入出力用ファイルを指定したファイル名によってオープンします。
fprintf	書式に従ってストリーム入出力用ファイルへデータを出力します。
fputc	ストリーム入出力用ファイルへ 1 文字出力します。

fputs	ストリーム入出力用ファイルへ文字列を出力します。
fread	ストリーム入出力用ファイルから指定した記憶域にデータを入力します。
freopen	現在オープンされているストリーム入出力用ファイルをクローズし、新しいファイルを指定したフイル名で再オープンします。
fscanf	ストリーム入出力用ファイルからデータを入力し、書式に従って変換します。
fseek	ストリーム入出力用ファイルの現在の読み書き位置を移動します。
fsetpos	ファイルストリーム上の位置を変更します。
ftell	ストリーム入出力用ファイルの現在の読み書き位置を求めます。
fwrite	記憶域からストリーム入出力用ファイルへデータを出力します。
getc	ストリーム入出力用ファイルから 1 文字入力します。
getchar	標準入力 (stdin) から 1 文字入力します。
gets	標準入力 (stdin) から文字列を入力します。
perror	標準エラー出力 (stderr) に、エラー番号に対応したエラーメッセージを出力します。
printf	データを書式に従って変換し、標準出力 (stdout) へ出力します。
putc	ストリーム入出力ファイルへ 1 文字出力します。
putchar	標準出力 (stdout) へ 1 文字出力します。
puts	標準出力 (stdout) へ文字列を出力します。
remove	ファイルを削除します。
ename	ファイル名を変更します。
rewind	ストリーム入出力用ファイルの現在の読み書き位置をファイルの先頭に移動します。
scanf	標準入力 (stdin) からデータを入力し、書式に従って変換します。
setbuf	ストリーム入出力用のバッファをユーザープログラム側で定義して設定します。
setvbuf	ストリーム入出力用のバッファをユーザープログラム側で定義し、さらにバッファの使用方法を設定します。
sprintf	データを書式に従って変換し、指定した領域へ出力します。
sscanf	指定した記憶域からデータを入力し、書式に従って変換します。

tmpfile	テンポラリファイルを生成します。
tmpnam	既存しないテンポラリファイル名を生成します。
ungetc	ストリーム入出力用ファイルへ１文字を戻します。
vfprintf	可変個のパラメータリストを書式に従って、指定したストリーム入出力用ファイルへ出力します。
vprintf	可変個のパラメータリストを書式に従って、標準出力へ出力します。
vsprintf	可変個のパラメータリストを書式に従って、指定した記憶域へ出力します。

マクロ名	説明
_IOFBF	バッファ領域の使用方法として、入出力処理はすべてバッファ領域を使用することを示しています。
_IOLBF	バッファ領域の使用方法として、入出力処理は行単位でバッファ領域を使用することを示しています。
_IONBF	バッファ領域の使用方法として、入出力処理はバッファ領域を使用しないことを示しています。
BUFSIZ	入出力処理において必要とするバッファの大きさを示しています。
EOF	ファイルの終わり (end of file) すなわちファイルからそれ以上の入力が無いことを示す文字です。
L_tmpnam	tmpnam 関数によって生成される一時ファイル名の文字列を格納するのに十分な大きさの配列のサイズを示しています。
SEEK_CUR	ファイルの現在の読み書き位置を現在の位置からのオフセットに移すことを示しています。
SEEK_END	ファイルの現在の読み書き位置をファイルの終了位置からのオフセットに移すことを示しています。
SEEK_SET	ファイルの現在の読み書き位置をファイルの先頭位置からのオフセットに移すことを示しています。
TMP_MAX	tmpnam 関数によって生成される一意なファイル名の個数の最大値を示します。
FOPEN_MAX	fopen 関数が一度にオープンできるファイルの最大数を示します。ただし、その最大数は stdio、stdout、stderr を含みます。
FILENAME_MAX	オープンできるファイル名の最大長を示します。

stderr	標準エラー出力ファイルに対応のファイルポインタを示します。
stdin	標準入力ファイルに対するファイルポインタを示します。
stdout	標準出力ファイルに対するファイルポインタを示します。

データ型	概要
FILE	ストリーム入出力処理で必要とするバッファへのポインタやエラー指示子、終了指示子などの各種制御情報を保存しておく構造体の型を示しています。
fpos_t	fsetpos、fgetpos 関数でファイルの位置を表すのに用います。

2.2.13 stdlib.h

標準的処理 (メモリ管理、終了処理) 関数のプロトタイプ宣言、および、標準的処理関数に必要なマクロ定義とデータ型の宣言を行っています。宣言している関数、定義しているマクロ、および、宣言しているデータ型は以下のとおりです。

関数	概要
abort	プログラムの実行を強制終了します。
abs	int 型整数の絶対値を計算します。
atexit	プログラムが正常終了した時に呼び出される関数を登録します。
atof	数を表現する文字列を double 型の浮動小数点数値に変換します。
atoi	10 進数を表現する文字列を int 型の整数値に変換します。
atol	10 進数を表現する文字列を long 型の整数値に変換します。
bsearch	２分割検索を行います。
calloc	記憶域を割り当てて、すべての割り当てられた記憶域を 0 によって初期化します。
div	int 型整数の除算の商と余りを計算します。
exit	プログラムを終ケします。
free	指定された記憶域を開放します。
getenv	環境変数の内容を取得します。
labs	long 型整数の絶対値を計算します。
ldiv	long 型整数の除算の商と余りを計算します。

malloc	記憶域を割り当てます。
mblen	マルチバイト文字のバイト数を求めます。
mbstowcs	マルチバイト文字列をワイド文字列に変換します。
mbtowc	マルチバイト文字をワイド文字に変換します。
qsort	ソートを行います。
rand	0 から RAND_MAX の間の擬似乱数整数を生成します。
realloc	記憶域の大きさを指定された大きさに変更します。
srand	rand 関数で生成する擬似乱数数列の初期値を設定します。
strtod	数を表現する文字列を double 型の浮動小数点数値に変換します。
strtol	数を表現する文字列を long 型の整数値に変換します。
strtoul	数を表現する文字列を unsigned long 型の整数値に変換します。
system	コマンド文字列をコマンドプロセッサによって実行されるホスト環境に渡します。
wcstombs	ワイド文字列をマルチバイト文字列に変換します。
wctomb	ワイド文字をマルチバイト文字に変換します。

マクロ名	説明
RAND_MAX	rand 関数において生成する擬似乱数整数の最大値を示しています。

データ型	概要
div_t	div 関数のリターン値の構造体の型を示しています。
ldiv_t	ldiv 関数のリターン値の構造体の型を示しています。

2.2.14 string.h

文字列操作関数とメモリ操作関数のプロトタイプ宣言を行っています。宣言している関数以下のとおりです。

関数	概要

memchr	記憶域において、指定された文字が最初に現われる位置を検索します。
memcmp	二つの記憶域を比較します。
memcpy	複写元の記憶域の内容を指定した大きさ分、複写先の記憶域に複写します。
memmove	複写元の記憶域の内容を指定した大きさ分、複写先の記憶域に移動します。
memset	記憶域の先頭から指定された文字を指定された文字数分設定します。
strcat	文字列の後に、文字列を連結します。
strchr	文字列において、指定された 1 文字が最初に現われる位置を検索します。
strcmp	二つの文字列を比較します。
strcoll	現在のロケールに基づいて指定された 2 つの文字列を比較します。
strcpy	複写元の文字列の内容を、複写先の記憶域に null 文字も含めて複写します。
strcspn	文字列において、指定外の文字が先頭から何文字続くかを求めます。
strerror	エラーメッセージを返します。
strlen	文字列の長さを計算します。
strncat	文字列の後に、文字列を指定した文字数分連結します。
strncmp	二つの文字列を指定された文字数分まで比較します。
strncpy	複写元の文字列を指定された文字数分、複写先の記憶域に複写します。
strpbrk	文字列において、指定された文字が最初に現われる位置を検索します。
strrchr	文字列において、指定された文字が最後に現われる位置を検索します。
strspn	文字列において、指定された文字が先頭から何文字続くかを求めます。
strstr	文字列において、別に指定した文字列が最初に現われる位置を検索します。

strtok	文字列をいくつかのトークン (字句) に切り分けます。
strxfrm	現在のロケールに基づいて文字列を変換します。

2.2.15 time.h

日付・時刻操作関数のプロトタイプ宣言とマクロ定義を行っています。宣言している関数以下のとおりです。

関数	概要
asctime	tm 型構造体で示される日時をテキスト文字列の形式に変換します。
clock	プログラムの実行時間を求めます。
ctime	time_t 型の暦時間をテキスト文字列の形式に変換します。
difftime	指定された 2 つの時間差を求めます。
gmtime	暦時間を世界標準時に変換します。
localtime	暦時間を現在の現地時間に変換します。
mktime	tm 型構造体で示される日時を暦時間に変換します。
strftime	tm 型構造体で示される日時を書式に従って変換します。
time	現在の暦時間を求めます。

3 C 標準ライブラリ関数詳細

3.1 assert.h – libc.a

3.1.1 assert – プログラム中に診断機能を付加

書式

```
#include <assert.h>
#include <stdio.h>
void assert(int expression);
int expression; /* 評価する式 */
```

戻り値

なし

解説

assert関数はマクロとして定義されており、*expression* が偽 (0) のとき、診断メッセージを標準エラーファイルに出力し、その後 abort関数を起動します。また、*expression* が真のときは何も行いません。assert関数は通常、プログラムの論理的なエラーの検出に用い、*expression* にはプログラムが正常に動作した場合だけ真になるような条件式を指定します。プログラムのデバッグ後 assertの記述をプログラムから削除する場合、<assert.h>を取り込む前にマクロ NDEBUG を#define文で定義してください(#define NDEBUG)。assertというマクロ名に対して#undef文を使用すると、それ以降の assert関数の呼び出し効果は保証されません。

3.2 ctype.h – libc.a

3.2.1 isalnum – 英字または 10 進数字かどうか判定

書式

```
#include <ctype.h>
int isalnum(int c);
int c; /* 判定する文字 */
```

戻り値

'0 以外の値'
: *c* が英数字 ('A'-'Z','a'-'z','0'-'9')

'0'
: *c* が英数字以外

解説

c が英数字ならば 0 以外の値を返し、英数字でなければ 0 を返します。英数字とは以下の文字を指します。

- 英大文字 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- 英小文字 a b c d e f g h i j k l m n o p q r s t u v w x y z
- 10 進数字 0 1 2 3 4 5 6 7 8 9

注意
c の値が unsigned char型で表現できる範囲に含まれず、かつ EOF でない場合、この関数の動作は保証されません。

3.2.2 isalpha – 英字かどうか判定

書式

```
#include <ctype.h>
int isalpha(int c);
int c; /* 判定する文字 */
```

戻り値

‘0 以外の値’
c が英字 (’A’-’Z’,’a’-’z’)

‘0’ *c* が英字以外

解説
c が英字ならば 0 以外の値を返し、英字でなければ 0 を返します。英字とは以下の文字を指します。

- 英大文字 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- 英小文字 a b c d e f g h i j k l m n o p q r s t u v w x y z

注意
c の値が unsigned char型で表現できる範囲に含まれず、かつ EOF でない場合、この関数の動作は保証されません。

3.2.3 iscntrl – 制御文字かどうか判定

書式

```
#include <ctype.h>
int iscntrl(int c);
int c; /* 判定する文字 */
```

戻り値

‘0 以外の値’
c が制御文字

‘0’ *c* が制御文字以外

解説
c が制御文字ならば 0 以外の値を返し、制御文字でなければ 0 を返します。制御文字とは印字文字以外の文字を指します。印字文字とは、ディスプレイに表示可能な文字で、ASCII コード 0x20〜0x7E に対応するものです。

注意
c の値が unsigned char型で表現できる範囲に含まれず、かつ EOF でない場合、この関数の動作は保証されません。

3.2.4 isdigit – 10 進数字かどうか判定

書式

```
#include <ctype.h>
int isdigit(int c);
int c; /* 判定する文字 */
```

戻り値

'0 以外の値'
　　c が 10 進数字

'0'　　c が 10 進数字以外

解説

c が 10 進数字ならば 0 以外の値を返し、10 進数字でなければ 0 を返します。10 進数字とは以下の文字を指します。

- 10 進数字 0 1 2 3 4 5 6 7 8 9

注意

c の値が unsigned char型で表現できる範囲に含まれず、かつ EOF でない場合、この関数の動作は保証されません。

3.2.5 isgraph – 空白を除く印字文字かどうか判定

書式

```
#include <ctype.h>
int isgraph(int c);
int c; /* 判定する文字 */
```

戻り値

'0 以外の値'
　　c が「空白を除く印字文字」

'0'　　c が「空白を除く印字文字」でない

解説

c が空白を除く印字文字ならば 0 以外の値を返し、空白を除く印字文字でなければ 0 を返します。「空白を除く印字文字」とは、ディスプレイに表示可能な文字で、ASCII コード 0x21〜0x7E に対応するものです。

注意

c の値が unsigned char型で表現できる範囲に含まれず、かつ EOF でない場合、この関数の動作は保証されません。

3.2.6 islower – 英小文字かどうか判定

書式

```
#include <ctype.h>
int islower(int c);
int c; /* 判定する文字 */
```

戻り値

'0 以外の値'
　　c が英小文字

'0'　　c が英小文字以外

解説

c が英小文字ならば 0 以外の値を返し、英小文字でなければ 0 を返します。英小文字とは以下の文字を指します。

・ 英小文字 a b c d e f g h i j k l m n o p q r s t u v w x y z

注意

c の値が unsigned char型で表現できる範囲に含まれず、かつ EOF でない場合、この関数の動作は保証されません。

3.2.7 isprint – 空白を含む印字文字かどうか判定

書式

```
#include <ctype.h>
int isprint(int c);
int c; /* 判定する文字 */
```

戻り値

'0 以外の値'
　　c が空白を含む印字文字

'0'　　c が空白を含む印字文字以外

解説

c が空白を含む印字文字ならば 0 以外の値を返し、空白を含む印字文字でなければ 0 を返します。印字文字とは、ディスプレイに表示可能な文字で、ASCII コード 0x20〜0x7E に対応するものです。

注意

c の値が unsigned char型で表現できる範囲に含まれず、かつ EOF でない場合、この関数の動作は保証されません。

3.2.8 ispunct – 特殊文字かどうか判定

書式

```
#include <ctype.h>
int ispunct(int c);
int c; /* 判定する文字 */
```

戻り値

‘0 以外の値’
　　c が特殊文字

‘0’　　c が特殊文字以外

解説
c が特殊文字ならば 0 以外の値を返し、特殊文字でなければ 0 を返します。特殊文字とは、空白、英大文字、英小文字、10 進数字をそれぞれ除いた任意の印字文字です。

注意
c の値が unsigned char型で表現できる範囲に含まれず、かつ EOF でない場合、この関数の動作は保証されません。

3.2.9 isspace – 空白類文字かどうか判定

書式

```
#include <ctype.h>
int isspace(int c);
int c; /* 判定する文字 */
```

戻り値

‘0 以外の値’
　　c が空白類文字

‘0’　　c が空白類文字以外

解説
c が空白類文字ならば 0 以外の値を返し、空白類文字でなければ 0 を返します。空白類文字とは以下の文字を指します。

- 空白類文字　空白 ( )、書式送り (\f)、改行 (\n)、復帰 (\r)、水平タブ (\t)、垂直タブ (\v)

注意
c の値が unsigned char型で表現できる範囲に含まれず、かつ EOF でない場合、この関数の動作は保証されません。

3.2.10 isupper – 英大文字かどうか判定

書式

```
#include <ctype.h>
int isupper(int c);
int c; /* 判定する文字 */
```

戻り値

‘0 以外の値’
　　c が英大文字

‘0’　　c が英大文字以外

解説

c が英大文字ならば 0 以外の値を返し、英大文字でなければ 0 を返します。英大文字とは以下の文字を指します。

・ 英大文字 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

注意

c の値が unsigned char型で表現できる範囲に含まれず、かつ EOF でない場合、この関数の動作は保証されません。

3.2.11 isxdigit – 16 進数字かどうか判定

書式

```
#include <ctype.h>
int isxdigit(int c);
int c; /* 判定する文字 */
```

戻り値

'0 以外の値'
　　c が 16 進数字

'0'　　*c* が 16 進数字以外

解説

c が 16 進数字ならば 0 以外の値を返し、16 進数字でなければ 0 を返します。16 進数字とは以下の文字を指します。

・ 16 進数字 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f

注意

c の値が unsigned char型で表現できる範囲に含まれず、かつ EOF でない場合、この関数の動作は保証されません。

3.2.12 tolower – 英大文字を英小文字に変換

書式

```
#include <ctype.h>
int tolower(int c);
int c; /* 変換する文字 */
```

戻り値

'変換された文字'
　　引数として指定した文字 *c* が条件を満たす場合

'文字 *c*（ そのまま）'
　　引数として指定した文字 *c* が条件を満たさない場合

解説

tolower関数は、文字 *c* が英大文字のとき小文字に変換し、変換した文字を返します。そうでないとき、*c* を変換せずそのまま返します。

3.2.13 toupper – 英小文字を英大文字に変換

書式

```
#include <ctype.h>
int toupper(int c);
int c; /* 変換する文字 */
```

戻り値

'変換された文字'
　　引数として指定した文字 *c* が条件を満たす場合

'文字 *c*(そのまま) '
　　引数として指定した文字が *c* 条件を満たさない場合

解説

toupper関数は、文字 *c* が英小文字のとき大文字に変換し、変換した文字を返します。そうでないとき、*c* を変換せずそのまま返します。

3.3 locale.h – libc.a

3.3.1 setlocale – ロケール情報の設定、検索

書式

```
#include <locale.h>
char *setlocale(int category, const char *locale);
int category; /* 設定するロケール情報部門 */
const char *locale; /* 設定するロケール */
```

戻り値

'locale に設定可能な文字列へのポインタ'
　　正常終了(locale 引数が返されるわけではありません)

'NULL'　　*category*、*locale* のどちらも当てはまるものがない

解説

ロケールは"C"環境のみサポートしています。setlocale関数は、*category* で示されるロケール情報部門を *locale* で指定したロケールに設定します。*locale* が NULL のときは、現在のロケール情報を求めます。そのときのプログラムのロケール情報は変化しません。以下に *category* に指定できるマクロを示します。

'category'
　　意味

'LC_ALL'　すべてのロケール情報を設定、検索します。

'LC_COLLATE'
　　strcoll関数, strxfrm関数に影響を与える情報を設定、検索します。

'LC_CTYPE'
: isdigit関数と isxdigit関数を除くすべての文字操作関数とマルチバイトを扱う関数すべてに影響を与える情報を設定、検索します。

'LC_MONETARY'
: localeconv関数が返す通貨情報に影響を与える情報を設定、検索します。

'LC_NUMERIC'
: 入出力関数と文字列操作関数で使用される小数点、および localeconv関数が返す通貨情報以外の情報に影響を与える情報を設定、検索します。

'LC_TIME' strftime関数に影響を与える情報を設定、検索します。

3.3.2 localeconv – lconv 型構造体を初期化

書式

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

戻り値

初期化した lconv 型構造体へのポインタ

解説

localeconv関数は、lconv 型構造体を初期化します。

注意

ロケールは"C"環境のみサポートしています。localeconv関数を使用するためには、この関数が(関数内部で) 呼び出している_get_core,_rel_core関数を作成する必要があります

3.4 math.h – libm.a

3.4.1 acos – 浮動小数点数の逆余弦を計算

書式

```
#include <math.h>
double acos(double x);
double x; /* 逆余弦を求める浮動小数点数 */
```

戻り値

引数の計算値

解説

acos関数は引数 x の逆余弦を計算し、0 からπまでの範囲の値を返します。x の値は-1 以上 1 以下の範囲でなければなりません。x が-1 より小さい、もしくは 1 より大きいときは *errno* に EDOMの値を設定します。

3.4.2 asin – 浮動小数点数の逆正弦を計算

書式

```
#include <math.h>
double asin(double x);
double x; /* 逆正弦を求める浮動小数点数 */
```

戻り値

引数の計算値

解説

asin関数は引数 x の逆正弦を計算し、-π/2 からπ/2 までの範囲の値を返します。x の値は-1 以上 1 以下の範囲でなければなりません。x が-1 より小さい、もしくは 1 より大きいときは *errno* に EDOMの値を設定します。

3.4.3 atan – 浮動小数点数の逆正接を計算

書式

```
#include <math.h>
double atan(double x);
double x; /* 逆正接を求める浮動小数点数 */
```

戻り値

引数の計算値

解説

atan関数は x の逆正接を計算します。atan関数の値は 0 からπまでの範囲の値を返します。

3.4.4 atan2 – 浮動小数点数どうしを除算した結果の逆正接を計算

書式

```
#include <math.h>
double atan2(double y,  double x);
double y;
double x;
```

戻り値

引数の計算値

解説

atan2関数は y/x の逆正接を計算します。atan2関数は-πからπまでの範囲の値を返します。なお、atan2関数の結果は点 (x, y) と原点 (0,0) を通る直線と x 軸がなす角を示しています。atan2関数の両引数 x、y が共に 0 のとき、*errno* に EDOMの値を設定します。

3.4.5 cos – 浮動小数点数のラディアン値の余弦を計算

書式

```
#include <math.h>
double cos(double x);
double x; /* 浮動小数点数( ラディアン単位）*/
```

戻り値

引数の計算値

解説

cos関数は x の余弦を計算します。

3.4.6 sin – 浮動小数点数のラディアン値の正弦を計算

書式

```
#include <math.h>
double sin(double x);
double x; /* 浮動小数点数( ラディアン単位) */
```

戻り値

引数の計算値

解説

sin関数は x の正弦を計算します。

3.4.7 tan – 浮動小数点数のラディアン値の正接を計算

書式

```
#include <math.h>
double tan(double x);
double x; /* 浮動小数点数( ラディアン単位) */
```

戻り値

引数の計算値

解説

tan関数は x の正接を計算します。関数の計算結果がdouble型の値として表せないとき、*errno* にERANGEの値を設定します。また、計算結果がオーバーフローの場合、戻り値にHUGE_VAL(負の場合-HUGE_VAL）を返します。

3.4.8 cosh – 浮動小数点数の双曲線余弦を計算

書式

```
#include <math.h>
double cosh(double x);
doubel x; /* 浮動小数点数 */
```

戻り値

'引数の計算値'
: 正常終了

'HUGE_VAL'
: オーバーフロー時

解説

cosh関数は x の双曲線余弦を計算します。関数の計算結果がdouble型の値として表せないとき、 *errno* にERANGEの値を設定します。また、計算結果がオーバーフローした場合は、戻り値にHUGE_VALを返します。

3.4.9 sinh – 浮動小数点数の双曲線正弦を計算

書式

```
#include <math.h>
double sinh(double x);
double x; /* 浮動小数点数 */
```

戻り値

引数の計算値

解説

sinh関数は x の双曲線正弦を計算します。関数の計算結果が double型の値として表せないとき、*errno* に ERANGEの値を設定します。また、計算結果がオーバフローのときは、戻り値に HUGE_VAL(負の場合は-HUGE_VAL）を返します。

3.4.10 tanh – 浮動小数点数の双曲線正接を計算

書式

```
#include <math.h>
double tanh(double x);
double x; /* 浮動小数点数 */
```

戻り値

引数の計算値

解説

tanh関数は x の双曲線正接を計算します。また、計算結果がオーバーフローの場合、戻り値に HUGE_VAL (負の場合-HUGE_VAL）を返します。

3.4.11 exp – 浮動小数点数の指数関数を計算

書式

```
#include <math.h>
double exp(double x);
double x; /* 浮動小数点数 */
```

戻り値

指数関数 (ex) の値

解説

exp関数は x の指数関数 (ex) を計算し、その結果を返します。変換結果がオーバーフローの場合、*errno* に ERANGEの値を設定し、戻り値として HUGE_VALを返します。

3.4.12 frexp – 浮動小数点数を(0.5, 1.0）の値と２の累乗の積に分解

書式

```
#include <math.h>
double frexp(double x, int *e);
double x; /* 浮動小数点数 */
int e; /* 指数部 (整数値) を格納する記憶域へのポインタ */
```

戻り値

係数部 m の値

解説

frexp関数は、x=m*2n が成り立つような 2 の累乗部 (n) とその係数部 (m) を求めます。ここで、係数部 m の絶対値は 1.0 より小さく、0.5 以上の値となります。累乗値 n は、引数 e が指す記憶域に格納され、係数部 m は戻り値として返されます。なお、x が 0.0 のとき m と n は共に 0.0 になります。

3.4.13 ldexp – 浮動小数点数と 2 の累乗の乗算を計算

書式

```
#include <math.h>
double ldexp(double x, int e);
double x; /* 浮動小数点数 */
int e; /* 指数 (整数型) */
```

戻り値

x*2e の演算結果

解説

ldexp関数は、x*2e の値を計算し、結果を返します。演算結果がオーバーフローのときは、*errno* に ERANGE の値を設定し、戻り値に HUGE_VAL(負の場合は-HUGE_VAL）返します。

3.4.14 log – 浮動小数点数の自然対数を計算

書式

```
#include <math.h>
double log(double x);
double x; /* 浮動小数点数 */
```

戻り値

x の自然対数

解説

log関数は x の自然対数を計算し、その結果を返します。x の値が負のとき *errno* に EDOMの値を設定します。x の値が 0.0 のとき *errno* に ERANGEの値を設定します。

3.4.15 log10 – 浮動小数点数の 10 を底をする対数を計算

書式

```
#include <math.h>
double log10(double x);
double x; /* 浮動小数点数 */
```

戻り値

x の常用対数

解説

log10関数は x の常用対数を計算し、その結果を返します。x の値が負のとき *errno* に EDOM の値を設定します。x の値が 0.0 のとき *errno* に ERANGEの値を設定します。

3.4.16 modf – 浮動小数点数を整数部分と小数部分に分解

書式

```
#include <math.h>
double modf(double x, double *i);
double x; /* 浮動小数点数 */
double *i; /* 整数部分を格納する記憶域を指すポインタ */
```

戻り値

'x の小数部分'
: modf関数の戻り値として返します

'x の整数部'
: i が指す記憶領域に格納します

解説

modf関数は、x の値を小数部と整数部に分け、小数部を返します。整数部は i が指す記憶域に格納します。

3.4.17 pow – 浮動小数点数の累乗を計算

書式

```
#include <math.h>
double pow(double x, double y);
double x; /* 浮動小数点数 */
double y; /* 浮動小数点数 */
```

戻り値

'$x$$y$ の値'　正常終了

'HUGE_VAL'
: 計算結果がオーバーフロー

解説

pow関数は、x の y 乗の値を返します。x の値が 0.0 で、かつ y の値が 0.0 以下のとき、あるいは x の値が負で y の値が整数値でないとき、*errno* に EDOMを設定します。また、計算結果がオーバフローした場合、*errno* に ERANGE を設定します。

3.4.18 sqrt – 浮動小数点数の正の平方根を計算

書式

```
#include <math.h>
double sqrt(double x);
double x; /* 浮動小数点数 */
```

戻り値

x の正の平方根

解説

sqrt関数は、x の正の平方根を計算します。x の値が負のとき *errno* に EDOMの値を設定します。

3.4.19 ceil – 浮動小数点数の小数点以下を切り上げた整数値を算出

書式

```
#include <math.h>
double ceil(double x);
double x; /* 浮動小数点数 */
```

戻り値

計算結果

解説

ceil関数は、x の値より大きいまたは等しい、最小の整数値を double型の値として返します。

3.4.20 fabs – 浮動小数点数の絶対値を計算

書式

```
#include <math.h>
double fabs(double x);
double x; /* 浮動小数点数 */
```

戻り値

x の絶対値

解説

fabs関数は、x の絶対値を返します。

3.4.21 floor – 浮動小数点数の小数点以下を切り捨てた整数値算出

書式

```
#include <math.h>
double floor(double x);
double x; /* 浮動小数点数 */
```

戻り値

計算結果

解説

floor関数は、x の値を超えない範囲の整数の最大値を double型の値として返します。

3.4.22 fmod – 浮動小数点数どうしを除算した結果の余りを計算

書式

```
#include <math.h>
double fmod(double x, double y);
double x; /* 浮動小数点数 */
double y; /* 浮動小数点数 */
```

戻り値

x/y の余り

解説

fmod関数は、x/y の余りを算出し、それを double型で返します。戻り値(f とする）は被除数(x）と同符号で、f の絶対値は除数(y）の絶対値より小さく、次の式が成り立ちます。

x = y * i + f ※iはある整数値

y=0 の場合など、x/y の商を表現できない場合、処理結果は保証されません。

3.5 setjmp.h – libc.a

3.5.1 setjmp – 現在の実行環境を指定記憶域に退避

書式

```
#include <setjmp.h>
int setjmp(jmp_buf env);
jmp_buf env; /* 実行環境が記憶されている変数 */
```

戻り値

'0'　　setjmp関数が実行された。*env* には実行環境が記憶される。

'longjmp関数の第 2 引数 *ret*'
　　longjmp関数より制御が移された。ただし、longjmp関数の第 2 引数 *ret* が 0 の場合はこの戻り値は 1 となる。

解説

setjmp関数は、現在実行中の関数の実行環境を指定した記憶域に退避します。setjmp関数はlongjmp関数と組み合わせて用い、関数外へのグローバルなジャンプを実現することができます。通常の関数呼び出しやリターンの規約を用いずに、以前に呼び出したルーチンへエラー処理の実行制御を渡す場合等に使用します。setjmp関数を複雑な式から呼び出す場合、式の評価の途中結果等の現在の実行環境の一部が失われる可能性があります。setjmp関数はsetjmp関数の結果と定数式の比較という形態だけで使用し、複雑な式の中では呼び出さないようにしてください。

3.5.2 longjmp – 実行環境を回復し、setjmp 関数呼び出し位置に制御を移動

書式

```
#include <setjmp.h>
void longjmp(jmp_buf env, int ret);
jmp_buf env; /* 実行環境が記憶されている変数 */
int ret; /* setjmp 関数への戻り値 */
```

戻り値

なし

解説

longjmp関数は、同じプログラム中で最後に呼び出されたsetjmp関数によって退避された関数の実行環境を回復し、そのsetjmp関数を呼び出したプログラムの位置に制御を移します。このときlongjmp関数の *ret* がsetjmp関数の戻り値として返ります。longjmp関数はリターンしません。longjmp関数は、ジャンプ先のsetjmp関数を実行する前に実行された場合、あるいはsetjmp関数を呼び出した関数が既に return 文を実行している場合、その動作は保証されません。

3.6 signal.h – libc.a

3.6.1 signal – シグナル応答関数を設定

書式

```
#include <signal.h>
void (*signal(int sig, void (*func)(int)))(int);
int sig; /* シグナルの値 */
void (*func)(int); /* シグナルを検出したときに実行される関数 */
```

戻り値

'func の以前(最近) の値'
　　正常終了

'SIG_ERR'　エラー(errno に適切な値を返します)

解説

signal関数は、*sig* で指定したシグナルを検出したときに呼び出される処理ハンドラ(*func*) を設定します。funcで指定するハンドラには、ユーザーで定義する以外に以下の特殊な 2 つのマクロがあります。

'SIG_DFL'　デフォルトのシグナル処理を実行します。

'SIG_IGN'　シグナルを無視します。

注意

signal関数を使用する場合は、signal関数自身を作成する必要があります。

3.6.2 raise – プログラムに対してシグナルを送信

書式

```
#include <signal.h>
int raise(int sig);
int sig; /* 発生させるシグナル */
```

戻り値

'0'　　正常終了

'0 以外'　　エラー

解説

raise関数は、プログラムに対して *sig* で示すシグナルを送ります。

注意

raise関数を使用するためには、raise関数自身を作成する必要があります。

3.7 stdarg.h – libc.a

3.7.1 va_start(マクロ) – 可変個引数参照のための初期設定処理

書式

```
#include <stdarg.h>
void va_start(va_list ap, parmN);
va_list ap; /* 引数リストを指すポインタ */
parmN; /* 引数の並びの最右端の引数の識別子 */
```

戻り値

なし

解説

va_start、va_arg、va_endマクロによって、可変個の引数を持つ関数に対してその引数の参照を可能にします。va_startマクロは va_arg、va_endマクロで使用される ap の初期化を行います。$parmN$ には、外部関数定義における引数の並びの最右端の引数の識別子、即ち「, ...」の直前の識別子を指定します。可変個の名前のない引数を参照するためには、最初に va_start マクロを呼び出す必要があります。

3.7.2 va_arg(マクロ) – 可変個引数から順に引数を取り出す

書式

```
#include <stdarg.h>
type va_arg(va_list ap, type);
va_list ap; /* 引数リストを指すポインタ */
type; /* 戻り値の型 */
```

戻り値

現在の引数

解説

va_start、va_arg、va_endマクロによって、可変個の引数を持つ関数に対してその引数の参照を可能にします。第 1 引数に va_startマクロで初期化した va_list 型の変数 ap を指定し、第 2 引数に参照する引数の型を指定します。ap の値は va_arg を使用する度に更新され、結果として可変個の引数が順次本マクロの戻り値として返されます。

注意

$type$ に、型変換によってサイズが変わるような型を指定した場合、正しく引数を参照することができません。このような型を指定すると動作は保証されません。

3.7.3 va_end(マクロ) – 可変個引数の参照を終了

書式

```
#include <stdarg.h>
void va_end(va_list ap);
va_list ap; /* 引数リストを指すポインタ */
```

戻り値

なし

解説

va_start、va_arg、va_endマクロによって、可変個の引数を持つ関数に対してその引数の参照を可能にします。va_endマクロは引数への参照を終了させます。*ap* は va_startマクロで初期化された *ap* と同じでなければなりません。また、関数からの return 実行前に va_end マクロが呼び出されないときは、その関数の動作は保証されません。

3.8 stdio.h – libc.a

3.8.1 remove – ファイルを削除

書式

```
#include <stdio.h>
int remove(const char *filename);
const char *filename; /* 削除するファイル名 */
```

戻り値

'0'　　削除に成功した

'0 以外'　　エラー

解説

remove関数は、*filename* で指定したファイルを削除します。 *filename* で指定したファイルがオープンされていても削除します。

注意

remove関数を使用するためには、remove関数自身を作成する必要があります。また、オープンされているファイルを変更したり、削除できるかどうかは、作成されたremove関数に依存します。

3.8.2 rename – ファイル名を変更

書式

```
#include <stdio.h>
int rename(const char *old, const char *new);
const char *old; /* 元のファイル名 */
const char *new; /* 新しいファイル名 */
```

戻り値

'0'　　ファイル名が変更できた

'0 以外'　　エラー

解説

rename関数は、*old* で指定したファイル名を *new* で指定したファイル名に変更します。*old* で指定したファイルがオープンされていてもファイル名を変更します。*new* で指定したファイルが既に存在してもファイル名を変更します。

注意
rename関数を使用するためには、rename関数自身を作成する必要があります。また、オープンされているファイルを変更したり、削除できるかどうかは動作環境に依存します。

3.8.3 tmpfile – テンポラリファイルを生成

書式

```
#include <stdio.h>
FILE *tmpfile(void);
```

戻り値

'生成したファイルのストリームへのポインタ'
　　ファイルが生成できた

'NULL'　　ファイルが生成できなかった

解説
tmpfile関数は、テンポラリファイルを生成します。生成されたファイルは、バイナリファイルの更新モード (wb+) でオープンされ、ファイルをクローズしたりプログラムが終了したときに自動的に削除されます。

3.8.4 tmpnam – 既存しないテンポラリファイル名を生成

書式

```
#include <stdio.h>
char *tmpnam(char *s);
char *s; /* tmpnam が生成したファイル名を格納するバッファ */
```

戻り値

'生成したファイル名'
　　ファイルが生成できた

'NULL'　　tmpnam関数呼び出しが TMP_MAX 回を超えた

解説
tmpnam関数は、既存しないファイル名でテンポラリファイル名を生成し、s で示すバッファに生成したファイル名を格納します。s に NULL を指定した場合は、tmpnam内部のバッファに結果が書き込まれます。なお、このバッファは、tmpnamを呼ぶ度に書き換わります。tmpnam関数は TMP_MAX(stdio.h に定義）回まで呼び出せ、TMP_MAX 回を超えた場合は NULL を返します。s は、最低でも L_tmpnam(stdio.h に定義）サイズのバッファを確保していなければなりません。

3.8.5 fclose – ストリーム入出力用ファイルをクローズ

書式

```
#include <stdio.h>
int fclose(FILE *fp);
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

'0'　　正常終了

'EOF'　　エラー

解説

fclose関数は、ファイルポインタ *fp* の示すストリーム入出力用ファイルをクローズします。なお、ストリーム入出力用ファイルの出力ファイルがオープンされており、まだ出力されていないデータがバッファに残っている場合、それをファイルに出力してからクローズします。また、入出力用のバッファがシステムによって自動的に割り付けられていた場合は、それを解放します。さらに、ストリームが tmpfile関数でオープンされたものであれば対応するファイルを削除します。

3.8.6 fflush – ストリーム入出力用ファイルの内容をファイルへ出力

書式

```
#include <stdio.h>
int fflush(FILE *fp);
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

'0'　　正常終了

'EOF'　　エラー

解説

fflush関数は、*fp* で指定したストリーム入出力用ファイルが出力用にオープンされているとき、ストリームのバッファの未出力内容をファイルに出力します。また、入力用にオープンされているとき、ungetc関数の指定を無効にします。

3.8.7 fopen – ストリーム入出力用ファイルを指定ファイル名でオープン

書式

```
#include <stdio.h>
FILE *fopen(const char *fname, const char *mode);
const char *fname; /* ファイル名 */
const char *mode; /* ファイルアクセスモード */
```

戻り値

'オープンしたストリームへのポインタ'
　　正常終了

'NULL'　　エラー

解説

fopen関数は、引数 *fname* に指定したファイルをオープンします。引数 *mode* には、ファイルのアクセスモードを下表より選んで設定してください。

‘アクセスモード’
: 意味

‘"r"’
: ファイルを読み出し用にオープンします。

‘"w"’
: ファイルを書き込み用にオープンします。

‘"a"’
: ファイルを追加用にオープンします。

‘"r+"’
: ファイルを読み出し用かつ更新用にオープンします。

‘"w+"’
: ファイルを書き込み用かつ更新用にオープンします。

‘"a+"’
: ファイルを追加用かつ更新用にオープンします。

ファイルはテキストモードでオープンされますが、各アクセスモードの後にbを付けることによってバイナリモードでオープンできます (例： r+b)。書き込みモードおよび追加モードで、存在しないファイルをオープンしようとした場合は、可能な限り新しいファイルを作成します。既存のファイルを書き込みモードでオープンした場合、ファイルの先頭から書き込みが行われ、以前の内容は消去されます。追加モードでオープンした場合、そのファイルの終わりの位置から書き込み処理が実行されます。更新モードでオープンした場合、そのファイルに対して入出力処理が可能になります。ただし、出力処理後に fflush、fseek、rewind 関数が実行されることなしに入力処理を続けることはできません。また、同様に入力処理後、上記関数を実行せずに出力処理を続けることはできません。

3.8.8 freopen – オープン中のファイルをクローズ後、新ファイル名で再オープン

書式

```
#include <stdio.h>
FILE *freopen(const char *fname, const char *mode, FILE *fp);
const char *fname; /* ファイル名 */
const char *mode; /* ファイルアクセスモード */
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

‘オープンしたストリームへのポインタ’
: 正常終了

‘NULL ポインタ’
: エラー

解説

freopen関数は、現在オープンされているストリーム入出力用ファイル *fp* をクローズし、同じ *fp* の指す FILE 構造体を再使用して新しいファイル *fname* をストリーム入出力用にオープンします。*mode* にはファイルのアクセスモードを、fopen関数で使用するアクセスモードの中から選んで設定してください。

3.8.9 setbuf – ストリームバッファをユーザー定義

書式

```
#include <stdio.h>
void setbuf(FILE *fp, char *buf);
FILE *fp; /* FILE 構造体へのポインタ */
char *buf; /* バッファへのポインタ */
```

戻り値

なし

解説

setbuf関数は、*fp* の示すストリーム入出力用ファイルのバッファを、デフォルトのバッファ[4] から、*buf* の指す記憶域に変更します。バッファサイズは変わりません。バッファサイズも変更する場合は setvbuf関数を使用してください。*buf* に NULL を設定すると、バッファは使用されません(バッファリングされない)。これは setvbuf関数の第 3 引数 *type*(バッファの管理方式) に_IONBF を指定した場合と同じです。

3.8.10 setvbuf – ストリームバッファをユーザー定義

書式

```
#include <stdio.h>
int setvbuf(FILE *fp, char *buf, int type, size_t size);
FILE *fp; /* FILE 構造体へのポインタ */
char *buf; /* バッファへのポインタ */
int type; /* バッファの管理方式 */
size_t size; /* バッファサイズ */
```

戻り値

'0'	新しいバッファが割り当てられた
'0 以外'	エラー(引数が不適切)

解説

setvbuf関数は、*fp* の示すストリーム入出力用ファイルのバッファを、デフォルトのバッファ[5] から、*buf* の指す記憶域に変更します。また、バッファサイズを *size* に、バッファの管理方式を *type* に変更します。管理方式は以下に示す 3 通りに設定できます。

'type'	意味
'_IOFBF'	フルバッファリング。入出力処理はすべてバッファを使用して行います。バッファがいっぱいになった場合かフラッシュ処理された場合にデータがバッファから取り出されます。

[4] fopenまたは freopen関数によるファイルオープン時に自動的に割り当てられるバッファ。デフォルトのバッファサイズ(BUFSIZ) は stdio.h で定義されています。

[5] fopenまたは freopen関数によるファイルオープン時に自動的に割り当てられるバッファ。デフォルトのバッファサイズ(BUFSIZ) は stdio.h で定義されています。

‘_IOLBF’　行バッファリング。入出力処理は行単位でバッファを使用して行います。バッファがいっぱいになった場合、改行文字が書き込まれた場合、またはフラッシュ処理された場合にデータがバッファから取り出されます。

‘_IONBF’　バッファリングなし。入出力処理はバッファを使用せず、ストリームに読み書きする単位で行います。

注意

setvbuf関数は、ストリーム入出力用ファイルがオープンされてから入出力処理が行われるまでの間で使用してください。また、ファイルをクローズする前にバッファを解放してはいけません。

3.8.11 fprintf – 書式に従ってストリーム入出力用ファイルへデータを出力

書式

```
#include <stdio.h>
int fprintf(FILE *fp, const char *control, ...);
FILE *fp; /* FILE 構造体へのポインタ */
const char *control; /* 書式を示す文字列へのポインタ */
...; /* 出力データを示す可変個引数列 */
```

戻り値

‘出力した文字数’
　正常終了

‘負の値’　エラー

解説

fprintf関数は、書式 *control* に従って、可変個引数列で示す出力データを変換および編集し、ファイルへのポインタ *fp* の指すストリーム入出力用ファイルへ出力します。以下より、書式の詳細について示します(*printf*、*sprintf*、*vfprintf*、*vprintf*、*vsprintf* 関数で扱う書式もこれと同様です)。

3.8.11.1 書式を示す文字列の構成

書式を示す文字列(*control* で指定する）は、通常文字列と変換仕様文字列から構成されます。変換仕様文字列の数と出現順は、可変個引数列で列挙する引数の数と順番に対応しています。

■通常文字列
　%で始まる文字列(変換仕様文字列）でないもの。そのまま出力されます。
　　例:　`fprintf(fp,"data=%02d", a);`　‘data=’部分が通常文字

■変換仕様文字列
　%で始まる文字列。可変個引数列の各引数(以下、引数）の変換方法を指定します。文字列は%を先頭に、以下の変換指定要素から必要なものを組み合わせて構成します。
　– フラグ

- フィールド幅
- 精度
- 引数のサイズ指定
- 変換指定子

例: `fprintf(fp,"data=%02d", a);` ‘%02d’部分が変換仕様。a を変換。

3.8.11.2 変換仕様文字列の形式

変換仕様文字列は次の形式で各要素を指定します([ ]:省略可能)

%[フラグ...][フィールド幅][.[精度]][引数のサイズ指定] 変換指定子

例: %02d ——— %+フラグ(0) +フィールド幅(2) +変換指定子(d)

各要素は連続して記述します(スペースで区切らない)。変換仕様文字列に対する引数がない場合(可変個引数列がない、または、引数の数が足りない) の動作は保証されません。変換仕様文字列より引数の数が多い場合、余分な引数はすべて無視されます。

3.8.11.3 変換仕様文字列の各要素の詳細

変換仕様文字列の各要素について、その機能と指定方法を以下に示します。

■フラグ 符号を付けるなど、出力するデータに対する修飾を指定します。指定できるフラグの種類とその意味を以下に示します。

‘フラグ’	意味
‘—’	左詰め。変換したデータの文字数が指定したフィールド幅より少ないとき、データをフィールド内で左詰めにして出力する。このフラグがないときは右詰めとなる。
‘+’	符号付加。 符号付きのデータに変換するとき、データの符号に従って、変換したデータの先頭に’+’ あるいは’-’ 符号を付ける。
‘空白’	符号なし。符号付きのデータの変換において、変換したデータの先頭に符号が付かないとき、そのデータの先頭に空白を付ける。上記の’+’ と共に指定すると、本フラグは無視される。
‘#’	修飾。変換指定子の種類に従って、以下のように変換後のデータに修飾を行う。

‘変換指定子’	修飾
‘c,d,i,s,u’	本フラグは無視される。
‘o’	変換したデータが 0 でない場合、先頭に 0 を付ける。
‘x,X’	変換したデータが 0 でない場合、先頭に 0x(X 変換の場合は 0X) を付ける。

‘e,E,f,g,G’
変換したデータに小数点以下がないときでも小数点を出力する。また、g,G の場合、変換後のデータの後に付く 0 は取り除かない。

‘0’ 変換指定子が d、i、o、u、x、X、e、E、f、g、G の場合、左詰めの空白がなくなるよう、数値の左側を複数の 0 で埋める。フラグ’ −’ と同時指定した場合、あるいは、変換指定子が d、i、o、u、x、X の場合、精度が指定されると本フラグは無視される。

■フィールド幅
変換したデータの出力文字数を、10 進数または*(アスタリスク）で指定します。変換したデータの出力文字数がフィールド幅より少ないときは、フィールド幅までそのデータの先頭に空白が付けられます。ただし、フラグ’ −’ を指定した場合は、データの後に空白が付けられます。変換したデータの出力文字数がフィールド幅より大きいときは、フィールド幅が変換結果を出力できる幅に拡張されます。フラグ’0’ を指定した場合は、出力するデータの先頭には空白ではなく文字’0’ が付けられます。

■精度 変換指定子の種類に従って変換したデータの精度を指定します。指定は、ピリオド (.) の後に 10 進数または*(アスタリスク）を続ける形式で行います。ピリオドの後に数字がない場合、0 を指定したものと仮定します。精度を指定した結果、フィールド幅の指定との間に矛盾が生じれば、フィールド幅の指定を無効とします。変換指定子の種類とそのときの精度指定の意味を以下に示します。

‘変換指定子’
精度指定

‘d,i,o,u,x,X’
変換したデータの最小の桁数を示す。

‘e,E,f’ 変換したデータの小数点以下の桁数を示す。

‘g,G’ 変換したデータの最大有効桁数を示す。

‘s’ 印字される最大文字数を示す。

■フィールド幅あるいは精度に対する指定
フィールド幅と精度指定の値には、*(アスタリスク）が指定できます。*を指定すると、その変換仕様に対応する可変個引数の値が、フィールド幅あるいは精度指定の値として使用されます。フィールド幅に対する可変個引数の値が負の場合、正のフィールド幅に-(マイナス）フラグが指定されたと解釈します。また、精度に対する可変個引数の値が負の場合、精度が省略されたものと解釈します。

■引数のサイズ指定
変換指定子が d、i、o、u、x、X、e、E、f、g、G、n の場合、対応する引数のサイズを、修飾子 h、l、L で指定できます。修飾子は変換指示子の直前に書きます。サイズ指定の種類とその意味を下表に示します。

サイズ指定	該当変換指定子	対応する引数に対して指定するサイズ
h	d,i,o,u,x,X	short int あるいは unsigned short int

	n	short int へのポインタ
l(エル)	d,i,o,u,x,X,n	long int あるいは unsigned long int
	n	long int へのポインタ
L	e,E,f,g,G	long double

変換指定子が d、i、o、u、x、X、e、E、f、g、G、n 以外の場合、本指定は無視されます。

■変換指定子

変換指定子は引数をどのような形式に変換するかを指定します。変換する引数が、構造体や配列型のときや、構造体や配列型を指すポインタのときは、動作は保証されません(ただし、s 変換で文字の配列を変換するとき、p 変換でポインタを変換するときを除きます)。以下に、変換指定子と変換方式を示します。ここに挙げていない文字を変換指定子として指定した場合、動作は保証されません。なお、「 変換対象となる引数の型」は、h、l、L によるサイズ指定がないときのものです。

変換指定子	変換の方式	変換対象となる引数の型	精度に対する注意事項
d	int 型データを符号付き 10 進数の文字列に変換する。d 変換と i 変換は同仕様。	int	精度の指定は小数点以降の桁数を表す。変換後の文字数が精度の値より少ない場合、文字列の先頭に 0 が付く。精度を省略すると 1 が仮定される。0 の値を持つデータを、精度 0 と指定して変換し出力しようとしても、何も出力されない。
i	int 型データを符号付き 10 進数の文字列に変換する。d 変換と i 変換は同仕様。	int	精度の指定は小数点以降の桁数を表す。変換後の文字数が精度の値より少ない場合、文字列の先頭に 0 が付く。精度を省略すると 1 が仮定される。0 の値を持つデータを、精度 0 と指定して変換し出力しようとしても、何も出力されない。
o	unsigned int 型データを符号なし 8 進数の文字列に変換する。	unsigned int	精度の指定は小数点以降の桁数を表す。変換後の文字数が精度の値より少ない場合、文字列の先頭に 0 が付く。精度を省略すると 1 が仮定される。0 の値を持つデータを、精度 0 と指定して変換し出力しようとしても、何も出力されない。

u	unsigned int 型データを符号なし 10 進数の文字列に変換する。	unsigned int	精度の指定は小数点以降の桁数を表す。変換後の文字数が精度の値より少ない場合、文字列の先頭に 0 が付く。精度を省略すると 1 が仮定される。0 の値を持つデータを、精度 0 と指定して変換し出力しようとしても、何も出力されない。
x	unsigned int 型データを符号なし 16 進数に変換する。	unsigned int	精度の指定は小数点以降の桁数を表す。変換後の文字数が精度の値より少ない場合、文字列の先頭に 0 が付く。精度を省略すると 1 が仮定される。0 の値を持つデータを、精度 0 と指定して変換し出力しようとしても、何も出力されない。
X	unsigned int 型データを符号なし 16 進数に変換する。16 進文字には、A,B,C,D,E,F を用いる。	unsigned int	精度の指定は小数点以降の桁数を表す。変換後の文字数が精度の値より少ない場合、文字列の先頭に 0 が付く。精度を省略すると 1 が仮定される。0 の値を持つデータを、精度 0 と指定して変換し出力しようとしても、何も出力されない。
f	double 型データを [-]ddd.ddd の形式の 10 進数の文字列に変換する。	double	精度の指定は小数点以降の桁数を表す。小数点以降の文字が存在するときには、必ず小数点の前に一桁の数字が出力される。精度を省略すると 6 が仮定される。精度に 0 を指定すると、小数点と小数点以降の文字は出力されない。出力データは丸められる。

e	double 型データを [-]ddd.ddd+-ddd の形式の 10 進数の文字列に変換する。指数は最低 2 桁は出力される。	double	精度の指定は小数点以降の桁数を表す。変換した文字は、小数点の前に一桁の数字が出力され、小数点以降に精度に等しい桁数の数字が出力される。精度を省略すると 6 が仮定される。精度に 0 を指定すると、小数点以降の文字は出力されない。出力データは丸められる。
E	double 型データを [-]d.dddE+-ddd の形式の 10 進数の文字列に変換する。指数は最低 2 桁は出力される。	double	精度の指定は小数点以降の桁数を表す。変換した文字は、小数点の前に一桁の数字が出力され、小数点以降に精度に等しい桁数の数字が出力される。精度を省略すると 6 が仮定される。精度に 0 を指定すると、小数点以降の文字は出力されない。出力データは丸められる。
g	変換する値と有効桁数を指定する精度の値から、f 変換形式で出力するか、e 変換(あるいは E 変換）形式で出力するかを決め double 型データを出力する。変換されたデータの指数が、-4 より小さい、または、有効桁数を指定する精度以上の場合、e 変換(あるいは E 変換）形式に変換する。変換後、小数点以下の末尾の 0 は除去される。また、小数点以下に文字がなければ小数点も除去される。	double	精度の指定は、変換されたデータの最大有効桁数を示す。

G	変換する値と有効桁数を指定する精度の値から、f 変換形式で出力するか、e 変換(あるいは E 変換) 形式で出力するかを決め double 型データを出力する。変換されたデータの指数が、-4 より小さい、または、有効桁数を指定する精度以上の場合、e 変換(あるいは E 変換) 形式に変換する。変換後、小数点以下の末尾の 0 は除去される。また、小数点以下に文字がなければ小数点も除去される。	double	精度の指定は、変換されたデータの最大有効桁数を示す。
c	int 型データを unsigned char 型データとし、そのデータに対応する文字に変換する。	int	精度の指定は無効となる。
s	char 型へのポインタ型データが指す文字列を、文字列の終了を示す null 文字まで、あるいは、精度で指定された文字数分、出力する(ただし、null 文字は出力されない)。	char へのポインタ	精度の指定は、出力する文字数を示す。精度を省略すると、データが指す文字列の null 文字までの文字が出力される(ただし、null 文字は出力されない)。
p	データをポインタとして、コンパイラごとに定義された印字可能な文字列に変換する。	void へのポインタ	精度の指定は無効となる。
n(変換なし)	データは int 型へのポインタ型とみなされ、データが指すメモリ領域にいままで出力データの文字数を設定する。	int へのポインタ	精度の指定は無効となる。
%(変換なし)	%を出力する。	なし	精度の指定は無効となる。

3.8.12 fscanf – ストリームからデータを入力し、書式に従って変換

書式

```
#include <stdio.h>
int fscanf(FILE *fp, const char *control, ...);
FILE *fp; /* FILE 構造体へのポインタ( 入力データへのポインタ) */
const char *control; /* 書式を示す文字列へのポインタ */
...; /* 各入力データの格納先へのポインタ( 可変個引数列) */
```

戻り値

‘変換し代入を行った引数の数’
　　正常終了(この数に、%n 変換、*による代入抑止は含まれません)

‘EOF’　　最初の変換処理(%%は除く) が成功する前に入力データが終了した。または、エラーが発生した。

解説

fscanf関数は、ファイルポインタ *fp* の示すストリーム入出力用ファイルからテキストデータを入力し、書式 *control* に従って変換および編集し、その結果を可変個引数列で示す記憶域へ格納します。入力データ(fscanf関数の場合、ファイル内のテキスト) は、1 つ以上の空白、水平タブ('\t')、または改行文字('\n') によって複数のトークンに区切られます。また、空白類文字自体は読み飛ばされます。

例えば、入力データ" ABCD abcd"は、"ABCD"と"abcd"の 2 トークンと認識され、それぞれ変換後に可変個引数列で示す記憶域に格納されます(この場合、可変個引数列には 2 つの引数が必要です)。

以下より、書式の詳細について示します(scanf関数、sscanf 関数で扱う書式もこれと同様です)。

3.8.12.1 書式を示す文字列の構成

書式を示す文字列(*control* で指定する) は、通常文字列、変換仕様文字列から構成されます。変換仕様文字列に含まれる変換指定子の数と順番は、各入力データの格納先へのポインタを示している、可変個引数列で列挙する引数(以下、引数) の数と順番に対応しています。

■通常文字列

空白類文字でなく、かつ、%で始まる文字列(変換仕様文字列) でないもの。通常文字を指定すると、入力データ(fscanf関数の場合、ファイル内のテキスト) から、それと一致する文字が入力されますB 実際に入力されたデータ内に一致しない文字があれば、その文字は入力ストリームに残されます。

■変換仕様文字列

%で始まる文字列。入力データの変換方法を指定します。文字列は%を先頭に、以下の変換指定要素から必要なものを組み合わせて指定します。

- *
- フィールド幅
- 引数のサイズ指定
- 変換指定子

3.8.12.2 変換仕様文字列の形式

変換仕様文字列は%を先頭に次の形式で指定します([ ]:省略可能)。

%[*][フィールド幅][引数のサイズ指定] 変換指定子

例:　%2d%f ――― %+フィールド幅(2) +変換指定子(d) +%+変換指定子(f)

各要素は連続して記述します(スペースで区切らない)。変換仕様文字列に対して、可変個引数列の引数が足りない場合、動作は保証されません。変換仕様文字列より引数の数が多い場合、余分な引数はすべて無視されます。

3.8.12.3 変換仕様文字列の各要素の詳細

変換仕様文字列の各要素について、その機能と指定方法を以下に示します。

■ *(アスタリスク)
変換指定子の前に*を付けると、入力データ中の対応トークンは読み込まれますが、対応する引数への書き込みが抑制されます。

■フィールド幅
入力データの最大読み込み幅(文字数)を 10 進数で指定します。

■引数のサイズ指定
変換指定子が d、i、o、u、x、X、e、E、f、g、G、n の場合、対応する引数のサイズを、修飾子 h、l、L で指定できます。修飾子は変換指示子の直前に書きます。サイズ指定の種類とその意味を下表に示します。

サイズ指定	該当変換指定子	対応する引数に対して指定するサイズ
h	d,i,o,u,x,X	short int あるいは unsigned short int
	n	short int へのポインタ
l(エル)	d,i,o,u,x,X,n	long int あるいは unsigned long int
	n	long int へのポインタ
L	e,E,f,g,G	long double

変換指定子が d、i、o、u、x、X、e、E、f、g、G、n 以外の場合、本指定は無視されます。

■変換指定子
入力データをどのような形式に変換するかを指定します。fscanf関数は、変換指定子が c、[、n、%でない限り、変換前に空白類文字でない文字(この文字はフィールド幅の対象にはなりません)まで読み飛ばします。変換中に空白類文字を読み込んだ場合、変換の対象として許されていない文字を見つけると、その文字を読まずに残して処理を終了します。変換中に、指定されたフィールド幅に達した場合は処理を終了します。以下に、変換指定子と変換方式を示します。ここに挙げていない文字を変換指定子として指定した場合、動作は保証されません。なお、「対応する引数の型」は、h、l、L によるサイズ指定がないときのものです。

変換指定子	変換の方式	対応する引数の型
d	10 進数字の文字列を整数型データに変換する。	int へのポインタ
i	先頭に符号が付いている 10 進数字の文字列、あるいは最後に u(U) または l(L) が付いている 10 進数字の文字列を整数型データに変換する。先頭が 0x(あるいは 0X)で始まっている文字列は 16 進数字として解釈し、文字列を整数型デ ータに変換する。先頭が 0 で始まっている文字列は 8 進数字として解釈し、文字列を整数型データに変換する。	int へのポインタ
o	8 進数字の文字列を整数型データに変換する。	unsigned int へのポインタ

u	符号なしの 10 進数字の文字列を整数型データに変換する。	unsigned int へのポインタ
x	16 進数字の文字列を整数型データに変換する。	unsigned int へのポインタ
X	x と X は同じ仕様。	unsigned int へのポインタ
s	空白、水平タブ、改行文字を読み出すまでをひとつの文字列として変換する。文字列の最後には null 文字を付加する(変換されたデータの格納域には、null 文字を付加した結果の文字列が格納できるサイズが必要)。	char へのポインタ
c	1 文字を入力する。このとき、入力する文字が空白類文字であっても読み飛ばされない。フィールド幅が指定されている場合、その指定分の文字が読み込まれる。したがって、このとき、変換したデータを格納する記憶域は、指定分の大きさが必要。	char へのポインタ
e	浮動小数点数を示す文字列を浮動小数点型データに変換する。入力引数の形式は strtod関数で表現できる浮動小数点数。	float へのポインタ
E	浮動小数点数を示す文字列を浮動小数点型データに変換する。入力引数の形式は strtod関数で表現できる浮動小数点数。	float へのポインタ
f	浮動小数点数を示す文字列を浮動小数点型データに変換する。入力引数の形式は strtod関数で表現できる浮動小数点数。	float へのポインタ
g	浮動小数点数を示す文字列を浮動小数点型データに変換する。入力引数の形式は strtod関数で表現できる浮動小数点数。	float へのポインタ
G	浮動小数点数を示す文字列を浮動小数点型データに変換する。入力引数の形式は strtod関数で表現できる浮動小数点数。	float へのポインタ
p	fprintf関数において、p 変換で変換される形式の文字列をポインタ型データに変換する。	void へのポインタ
n	データの入力は行わず、いままでに入力したデータの文字数が設定される。	int へのポインタ

[	[abcd]、[a-z]、[^abcd] のように、[と] で囲んだ文字セット指定する(この文字セットはスキャンセット (scan set) といい、本関数で読み込むべき文字の集合を定義するもの)。スキャンセットの最初が^でなければ、スキャンセット内に含まれない文字が最初に出現する前までが有効文字列として読み込まれる(すなわち有効文字はスキャンセット内の文字のみを含む)。スキャンセットの最初が^ならばスキャンセット内に含まれる文字が最初に出現する前までが有効文字列として読み込まれる(すなわち有効文字はスキャンセット以外の文字のみを含む)。読み込まれた有効文字列の最後には自動的に null 文字が付加される (対応する引数の指す領域、すなわち読み込まれた文字列を格納する領域には null 文字も含めて文字列が格納できるだけのサイズが必要)。	char へのポインタ
%	%を読み出す(変換も引数への書き込みもなし)。	引数なし

3.8.13 printf – データを書式に従って変換し、標準出力 (stdout) へ出力

書式

```
#include <stdio.h>
int printf(const char *control,...);
const char *control; /* 書式を示す文字列へのポインタ */
...; /* 出力データを示す可変個引数列 */
```

戻り値

'出力した文字数'
　　正常終了

'負の値'　エラー

解説

printf関数は、書式 *control* に従って、可変引数列を変換および編集し、標準出力ファイル(stdout)へ出力します。書式 *control* の詳細は fprintf関数の解説を参照してください。

3.8.14 scanf – 標準入力 (stdin) データを書式に従って変換

書式

```
#include <stdio.h>
int scanf(const char *control, ...);
const char *control; /* 書式を示す文字列へのポインタ */
...;    /* 各入力データ格納先ポインタ( 可変個引数列 ) */
```

戻り値

'変換し代入を行った引数の数 正常終了(この数には、%n 変換、および*による代入抑止は含まれません)'

'EOF'　　最初の変換処理(%%は除く) が成功する前に入力データが終了した。またはエラーが発生した。

解説

scanf関数は、標準入力(stdin) からデータを入力し、書式 *control* に従って変換および編集し、その結果を可変個引数列の指す記憶域へ格納します。書式 *control* の詳細は fscanf関数の解説を参照してください。

3.8.15 sprintf – データを書式変換し、指定した領域へ出力

書式

```
#include <stdio.h>
int sprintf(char *s, const char *control, ...);
char *s; /* データを出力する記憶域へのポインタ */
const char *control; /* 書式を示す文字列へのポインタ */
... ; /* 出力データを示す可変個引数列 */
```

戻り値

出力した文字数

解説

sprintf関数は、書式 *control* に従って、可変個引数列で示す各データを変換および編集し、*s* の指す記憶域へ出力します。書式 *control* の詳細は、fprintf関数の解説を参照してください。

3.8.16 sscanf – 指定記憶域データを入力し、書式変換

書式

```
#include <stdio.h>
int sscanf(char s, const char *control, ...);
char s; /* 入力データを格納した記憶域へのポインタ */
const char control; /* 書式を示す文字列へのポインタ */
... ; /* 各入力格納先へのポインタ( 可変個引数列) */
```

戻り値

'変換し代入を行った引数の数'
　　正常終了(この数には、%n 変換,*による代入抑止は含まれません)

'EOF'　　最初の変換処理(%%は除く) が成功する前に入力データが終了した。またはエラーが発生した。

解説

sscanf関数は、*s* の指す記憶域からデータを入力し、書式 *control* に従って変換および編集し、その結果を可変個引数列で示す記憶域へ格納します。書式 *control* の詳細は fscanf関数の解説を参照してください。

3.8.17 vfprintf – 可変個引数を書式変換しストリームへ出力

書式

```
#include <stdarg.h>
#include <stdio.h>
int vfprintf(FILE  *fp, const char *control, va_list arg);
FILE *fp; /* FILE 構造体へのポインタ */
const char *control; /* 書式を示す文字列へのポインタ */
va_list arg; /* 引数並びを指すポインタ */
```

戻り値

'出力した文字数'
　　正常終了

'負の値'　　出力エラーが発生時

解説

vfprintf関数はストリーム入出力用ファイル *fp* に可変個の引数をフォーマットを指定して出力します。この関数は fprintf 関数と同じですが、vfprintf関数の場合、「 引数の並び(可変個引数列）」ではなく「 引数の並びへのポインタ」を受け取ります。vfprintf関数は出力した文字列の最後に null 文字を追加しますが、この null 文字は戻り値として数えません。*control* で指定する書式仕様は fprintf関数で取り扱う書式仕様と同じです。詳細は fprintf関数の仕様を参照してください。引数リストを示す *arg* は、va_startおよび va_arg マクロによって初期化されていなければなりません。なお、本関数は va_endマクロを呼び出しません。

3.8.18 vprintf – 可変個引数を書式変換し標準出力へ出力

書式

```
#include <stdarg.h>
#include <stdio.h>
int vprintf(const char *control, va_list arg);
const char *control; /* 書式を示す文字列へのポインタ */
va_list arg; /* 引数並びを指すポインタ */
```

戻り値

'出力した文字数'
　　正常終了

'負の値'　　出力エラーが発生時

解説

vprintf関数は標準出力(stdout）に可変個の引数をフォーマットを指定して出力します。この関数は printfと同じですが、vprintf関数の場合、「 引数の並び(可変個引数列）」ではなく「 引数の並びへのポインタ」を受け取ります。vprintf関数は出力した文字列の最後に null 文字を追加しますが、この null 文字は戻り値として数えません。*control* で指定する書式仕様は fprintf関数で取り扱う書式仕様と同じです。詳細は fprintf関数の仕様を参照してください。引数リストを示す *arg* は、va_startおよび va_arg マクロによって初期化されていなければなりません。なお、本関数は va_endマクロを呼び出しません。

3.8.19 vsprintf – 可変個引数を書式変換し指定記憶域へ出力

書式

```
#include <stdarg.h>
#include <stdio.h>
int vsprintf(char *s, const char *control, va_list arg);
char *s; /* データを出力する記憶域へのポインタ */
const char *control; /* 書式を示す文字列へのポインタ */
va_list arg; /* 引数並びを指すポインタ */
```

戻り値

出力した文字数

解説

vsprintf関数は記憶域 *s* に可変個の引数をフォーマットを指定して出力します。この関数はsprintfと同じですが、vsprintf関数関数の場合、「 引数の並び(可変個引数列）」ではなく「 引数の並びへのポインタ」を受け取ります。vsprintf関数は出力した文字列の最後に null 文字を追加しますが、この null 文字は戻り値として数えません。*control* で指定する書式仕様は、fprintf関数で取り扱う書式仕様と同じです。詳細は fprintf関数の仕様を参照してください。引数リストを示す *arg* は、va_startおよび va_argマクロによって初期化されていなければなりません。なお、本関数は va_endマクロを呼び出しません。

3.8.20 fgetc – ストリーム入出力用ファイルから１文字入力

書式

```
#include <stdio.h>
int fgetc(FILE *fp);
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

‘読み込んだ 1 文字’
　　正常終了

‘EOF’　　ファイル終端(EOF）の読み込み。または、読み込みエラー発生。

解説

fgetc関数は、ファイルポインタ *fp* が示すストリーム入出力用ファイルから 1 文字読み込み、ファイルのファイル位置指示子(読み書き位置を示す）を進め、読み込んだ文字を返します。ファイルの終端(EOF）を読み込んだ場合、ファイルのファイル終了指示子(ファイルが終了したか否かを示す）が設定され、本関数は EOF を返します。処理中に読み込みエラーが発生した場合、ファイルのエラー指示子(エラーが発生したか否かを示す）が設定され、本関数は EOF を返します。エラー指示子は ferror関数、ファイル終了指示子は feof 関数によって参照できます。

3.8.21 fgets – ストリーム入出力用ファイルから文字列を入力

書式

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *fp);
char *s; /* データの格納領域へのポインタ */
int n; /* 格納する文字数 */
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

'文字列を格納する記憶域へのポインタ s'
　　正常終了

'NULL'　　ファイル終端(EOF) の読み込み。または、読み込みエラー発生。※ s が指すデータ格納域の内容は、入力ファイルの終了時には変化しませんが、エラー発生時にはその内容は保証されません。

解説

fgets関数は、ファイルポインタ fp の示すストリーム入出力用ファイルから文字列を読み込み、ポインタ s の指す記憶域に格納します。n-1 文字、改行文字まで、あるいは、ファイルの終わりまでの文字を読み込み、その文字列の最後に null 文字を付加します。読み込んだ改行文字は、格納する文字列に含まれます。ファイルの終端(EOF) を読み込んだ場合、ファイルのファイル終了指示子(ファイルが終了したか否かを示す) が設定され、本関数は NULL を返します。処理中に読み込みエラーが発生した場合、ファイルのエラー指示子(エラーが発生したか否かを示す) が設定され、本関数は NULL を返します。エラー指示子は ferror関数、ファイル終了指示子は feof 関数によって参照できます。

3.8.22 fputc – ストリーム入出力用ファイルへ１文字出力

書式

```
#include <stdio.h>
int fputc(int c, FILE *fp);
int c; /* 出力する文字 */
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

'出力した文字 c'
　　正常終了

'EOF'　　書き込みエラー

解説

fputc関数は、ファイルポインタ fp の示すストリーム入出力用ファイルに文字 c を出力し、ファイルのファイル位置指示子(読み書き位置を示す) を進め、出力した文字を返します。処理中に書き込みエラーが発生した場合、出力先ファイルのエラー指示子(エラーが発生したか否かを示す) が設定され、本関数は EOF を返します。エラー指示子は ferror関数によって参照できます。

3.8.23 fputs – ストリーム入出力用ファイルへ文字列を出力

書式

```
#include <stdio.h>
int fputs(char *s, FILE *fp);
char *s; /* 出力する文字列へのポインタ */
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

'0'　　正常終了

'EOF'　　書き込みエラー

解説

fputs関数は、ファイルポインタ *fp* の示すストリーム入出力用ファイルに文字列 *s* を出力します(ただし文字列の最後の null 文字は出力されません)。書き込みが成功した場合、本関数は負でない値を返します。処理中に書き込みエラーが発生した場合、出力先ファイルのエラー指示子(エラーが発生したか否かを示す) が設定され、本関数は EOF を返します。エラー指示子は ferror関数によって参照できます。

3.8.24 getc – ストリームから 1 文字入力

書式

```
#include <stdio.h>
int getc(FILE *fp);
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

'読み込んだ 1 文字'
　　正常終了

'EOF'　　ファイル終端(EOF) の読み込み。または、読み込みエラー発生。

解説

getc関数は、ファイルポインタ *fp* の示すストリーム入出力用ファイルから 1 文字読み込み、ストリームのファイル位置指示子(読み書き位置を示す) を進め、読み込んだ文字を返します。ファイルの終端(EOF) を読み込んだ場合、ファイルのファイル終了指示子(ファイルが終了したか否かを示す) が設定され、本関数は EOF を返します。処理中に読み込みエラーが発生した場合、ファイルのエラー指示子(エラーが発生したか否かを示す) が設定され、本関数は EOF を返します。エラー指示子は ferror関数、ファイル終了指示子は feof 関数によって参照できます。

3.8.25 getchar – 標準入力(stdin) から 1 文字入力

書式

```
#include <stdio.h>
int getchar( void);
```

戻り値

‘読み込んだ 1 文字’
　　正常終了

‘EOF’　　ファイル終端(EOF）の読み込み。または、読み込みエラー発生。

解説

getchar関数は、getc(stdin)に相当します(標準入力 stdin から 1 文字読み込み、読み込んだ文字を返します)。ファイルの終端(EOF）を読み込んだ場合、ファイルのファイル終了指示子(ファイルが終了したか否かを示す）が設定され、本関数は EOF を返します。処理中に読み込みエラーが発生した場合、ファイルのエラー指示子(エラーが発生したか否かを示す）が設定され、本関数は EOF を返します。エラー指示子は ferror関数、ファイル終了指示子は feof 関数によって参照できます。

3.8.26 gets – 標準入力(stdin）から文字列を入力

書式

```
#include <stdio.h>
char *gets(char *s);
char *s; /* データの格納領域へのポインタ */
```

戻り値

‘文字列を格納する記憶域へのポインタ s’
　　正常終了

‘NULL’　　ファイル終端(EOF）の読み込み。または、読み込みエラー発生。

※ s が指すデータ格納域の内容は、標準入力ファイルの終了時には変化しませんが、エラー発生時にはその内容は保証されません。

解説

gets関数は、標準入力 (stdin) から 1 行分のデータを入力し、s が指す記憶域に格納します。1 行分のデータとは、入力開始から改行文字まで、または、ファイルの終端までの文字列をいいます。読み込んだ文字列の改行文字や EOF は捨てられ、最後に null 文字が付加されます。ファイルの終端(EOF）を読み込んだ場合、ファイルのファイル終了指示子(ファイルが終了したか否かを示す）が設定され、本関数は NULL を返します。処理中に読み込みエラーが発生した場合、ファイルのエラー指示子(エラーが発生したか否かを示す）が設定され、本関数は NULL を返します。エラー指示子は ferror関数、ファイル終了指示子は feof 関数によって参照できます。

3.8.27 putc – ストリーム入出力ファイルへ 1 文字出力

書式

```
#include <stdio.h>
int putc(int c, FILE *fp);
int c; /* 出力する文字 */
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

'出力した文字 c'
　　正常終了

'EOF'　　書き込みエラー

解説

putc関数は、ファイルポインタ fp の示すストリーム入出力用ファイルに文字 c を出力し、ファイルのファイル位置指示子(読み書き位置を示す) を進め、出力した文字を返します。処理中に書き込みエラーが発生した場合、出力先ファイルのエラー指示子(エラーが発生したか否かを示す) が設定され、本関数は EOF を返します。エラー指示子は ferror関数によって参照できます。

3.8.28 putchar – 標準出力(stdout) へ 1 文字出力

書式

```
#include <stdio.h>
int putchar(int c);
int c; /* 出力する文字 */
```

戻り値

'出力した文字 c'
　　正常終了

'EOF'　　書き込みエラー

解説

putchar関数は、putc(c,stdout)に相当します(標準出力 stdout へ文字 c を出力し、その文字を返します)。処理中に書き込みエラーが発生した場合、出力先ファイルのエラー指示子(エラーが発生したか否かを示す) が設定され、本関数は EOF を返します。エラー指示子は ferror関数によって参照できます。

3.8.29 puts – 標準出力(stdout) へ文字列を出力

書式

```
#include <stdio.h>
int puts(char *s);
char *s; /* 出力する文字列へのポインタ */
```

戻り値

'0'　　正常終了

'EOF'　　書き込みエラー

解説

fputs関数は、標準出力 (stdout) に文字列 s を出力します。このとき、文字列の最後の null 文字は、改行文字に置き換えられて出力されます。書き込みが成功した場合、本関数は負でない値を返します。処理中に書き込みエラーが発生した場合、出力先ファイルのエラー指示子(エラーが発生したか否かを示す) が設定され、本関数は EOF を返します。エラー指示子は ferror関数によって参照できます。

3.8.30 ungetc – ストリーム入出力用ファイルへ１文字戻す

書式

```
#include <stdio.h>
int ungetc(int c, FILE *fp);
int c; /* 戻す文字 */
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

'戻した文字 c'
　　正常終了(通常)

'EOF'　　エラー

解説

ungetc関数は、文字 c をファイルポインタ fp が示すストリーム入出力用ファイルに戻します。また、ここで戻された文字は、fflush、fseek、rewind関数を呼び出さなければ次回の入力データとなります。もしこれらの関数を呼び出すことなく ungetc関数を 2 回以上呼び出した場合、その動作は保証されません。なお、ungetc関数が実行されるとファイルに対する現在の位置指示子がひとつ戻されますが、この位置指示子が既にファイルの先頭に位置している場合、位置指示子は保証されません。

ファイルの終端(EOF) を読み込んだ場合、ファイルのファイル終了指示子(ファイルが終了したか否かを示す) が設定され、本関数は EOF を返します。処理中に読み込みエラーが発生した場合、ファイルのエラー指示子(エラーが発生したか否かを示す) が設定され、本関数は EOF を返します。エラー指示子は ferror関数、ファイル終了指示子は feof関数によって参照できます

3.8.31 fread – ストリーム入出力用ファイルから指定した記憶域にデータを入力

書式

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t n, FILE *fp);
void *ptr; /* データ格納領域へのポインタ */
size_t size; /* 1 メンバのバイト数 */
size_t n; /* 読み出すメンバの数 */
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

‘読み出しに成功したメンバ数(通常 n と同じ値)’
　　正常終了

‘n より小さい値’
　　ファイルの終了時やエラー発生時(ファイルの終了かエラー発生かの区別はferror、feof関数を用います)

解説

fread関数は、$size$ で指定したバイト数を 1 メンバとしたデータを n メンバ、ファイルポインタ fp が示すストリーム入出力用ファイルから読み込んで ptr が指す記憶域に格納します。$size$ もしくは n が 0 のとき、戻り値として 0 を返し、ptr の指す記憶域の内容は変化しません。また、エラーが発生した場合、またはメンバの途中までしか読み出せなかった場合、そのファイルの位置指示子の値は保証されません。

3.8.32 fwrite – 記憶域からストリームへデータを出力

書式

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fp);
const void *ptr; /* データ格納領域へのポインタ */
size_t size; /* 1 メンバのバイト数 */
size_t n; /* 書き込むメンバの数 */
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

‘実際に出力に成功したメンバの数(通常 n と同じ値)’
　　正常終了

‘n より小さい値’
　　エラー発生時

解説

fwrite関数は、$size$ で指定したバイト数を 1 メンバとしたデータを n メンバ、ptr の指す記憶域から読み出してファイルポインタ fp が示すストリーム入出力用ファイルに出力します。

3.8.33 fgetpos – 現在のファイルストリームの位置を求める

書式

```
#include <stdio.h>
int fgetpos(FILE *stream, fpos_t *pos);
FILE *stream; /* FILE 構造体へのポインタ */
fpos_t *pos; /* 現在のファイルストリームの位置 */
```

戻り値

‘0’　　正常終了

‘0 以外’　　エラー(errno にエラー番号を返します)

解説

fgetpos関数は、*stream* で指定されたファイルストリームの現在の位置を *pos* が指す記憶域に返します。

3.8.34 fseek – ストリーム入出力の現在の読み書き位置を移動

書式

```
#include <stdio.h>
int fseek(FILE *fp, long offset, int type);
FILE *fp; /* FILE 構造体へのポインタ */
long offset; /* type による指定位置からのオフセット */
int type; /* オフセットの種類 */
```

戻り値

'0'　　正常終了

'0 以外'　　エラー

解説

fseek関数は、ファイルポインタ *fp* が示す、ストリーム入出力用ファイルの現在の読み書き位置を、オフセットの種類 *type* で指定した場所から *offset* バイト先の位置に移動します。オフセットの種類は以下のとおりです。

type 引数	意味	offset 引数
SEEK_SET	ファイルの先頭	0、正の値
SEEK_CUR	現在の読み書き位置	負の値、0、正の値
SEEK_END	ファイルの最後	0、負の値

テキストファイルの場合、*type* は SEEK_SET でかつ、*offset* は 0 かそのファイルに対する ftell関数によって返された値でなければなりません。fseek関数を呼び出すことによって ungetc関数の効果はなくなります。

3.8.35 fsetpos – ストリーム上の位置を変更

書式

```
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *pos);
FILE *stream; /* FILE 構造体へのポインタ */
const fpos_t *pos; /* 変更するファイルストリームの位置 */
```

戻り値

'0'　　正常終了

'0 以外'　　エラー(errno にエラー番号を返します)

解説

fsetpos関数は、*stream* で指定されたファイルストリームの位置を *pos* が指す記憶域で指定した位置に変更します。*pos* は、fgetposで返された値を指定します。

3.8.36 ftell – ストリームの現在の読み書き位置を求める

書式

```
#include <stdio.h>
long ftell(FILE *fp);
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

ファイルの現在の読み書き位置

解説

ftell関数は、ファイルポインタ *fp* が示すストリーム入出力用ファイルの現在の読み書き位置を求め、戻り値として返します。

3.8.37 rewind – ストリームの現在の読み書き位置を先頭に移動

書式

```
#include <stdio.h>
void rewind(FILE *fp);
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

なし

解説

rewind関数は、ファイルポインタ *fp* が示すストリーム入出力用ファイルの現在の読み書き位置を、ファイルの先頭に移動します。rewind関数を呼び出すことによって、ungetc関数の効果はなくなります。

3.8.38 clearerr – ストリーム入出力用ファイルのエラー状態をクリア

書式

```
#include <stdio.h>
void clearerr(FILE *fp);
FILE *fp; /*  FILE 構造体へのポインタ  */
```

戻り値

なし

解説

clearerr関数は、ファイルポインタ *fp* が示すストリーム入出力用ファイルに対するエラー指示子と終了指示子をクリアします。エラー指示子、ファイル終了指示子とは、ストリームファイル毎に保持しているデータです。これらのデータはそれぞれ、エラーが発生したか否か、ファイルが終了したか否かを示し、それぞれferror、feof関数によって参照できます。ストリームファイルを扱う関数のうち、その戻り値だけではエラーの発生やファイルの終了の情報が得られない場合、これらのデータを用いてファイルの状態を調べます。

3.8.39 feof – ストリーム入出力用ファイルの終了判定

書式

```
#include <stdio.h>
int feof(FILE *fp);
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

'0'　　ファイルが終わりでないとき

'0 以外'　　ファイルが終わりのとき

解説

feof関数はファイルポインタ *fp* が示すストリーム入出力用ファイルが終了したか否かを判定します。feof関数はファイル終了指示子が設定されていればファイルが終わりと判断します。

3.8.40 ferror – ストリーム入出力用ファイルのエラー状態判定

書式

```
#include <stdio.h>
int ferror(FILE *fp);
FILE *fp; /* FILE 構造体へのポインタ */
```

戻り値

'0'　　ファイルがエラー状態でない

'0 以外'　　ファイルがエラー状態

解説

ferror関数は、ファイルポインタ *fp* が示すストリーム入出力用ファイルがエラー状態か否かを判定します。ferror関数は、エラー指示子が設定されているときエラー状態であると判断します。

3.8.41 perror – エラー番号に対応したエラーメッセージを出力

書式

```
#include <stdio.h>
void perror(const char *s);
const char *s; /* エラーメッセージへのポインタ */
```

戻り値

なし

解説

perror関数は、標準エラー出力 (stderr) へ *s* で示されるエラーメッセージと *errno* とを対応させ出力します。

3.9 stdlib.h – libc.a

3.9.1 atof – 数を表現する文字列を double 型の浮動小数点数値に変換

書式

```
#include <stdlib.h>
double atof(const char *nptr);
const char *nptr; /* 変換対象文字列へのポインタ */
```

戻り値

'変換された値'
　　正常終了

'HUGE_VALまたは-HUGE_VAL'
　　オーバーフロー時

'0'　　アンダーフロー時

解説

atof関数は、数を表現する文字列 *nptr* をそれぞれ double型の値に変換し、変換した値を返します。変換結果がオーバーフローあるいはアンダーフローの場合、*errno* に ERANGEの値を設定し、戻り値に HUGE_VAL(負の場合は-HUGE_VAL）あるいは 0 を返します。

3.9.2 atoi – 10 進数を表現する文字列を int 型の整数値に変換

書式

```
#include <stdlib.h>
int atoi(const char *nptr);
const char *nptr; /* 変換対象文字列へのポインタ */
```

戻り値

'変換された値'
　　正常終了

'INT_MAXまたは INT_MIN'
　　オーバーフロー時

解説

atoi関数は数を表現する文字列 *nptr* を int型の値に変換し、変換した値を返します。変換結果がオーバーフローの場合、*errno* に ERANGEの値を設定し、戻り値に INT_MAX(負の場合は INT_MIN）を返します。

3.9.3 atol – 10 進数を表現する文字列を long 型の整数値に変換

書式

```
#include <stdlib.h>
long atol(const char *nptr);
const char *nptr; /* 変換対象文字列へのポインタ */
```

戻り値

'変換された値'
　　正常終了

'LONG_MAXまたはLONG_MIN'
　　オーバーフロー時

解説

atol関数は数を表現する文字列 *nptr* を long型の値に変換し、変換した値を返します。変換結果がオーバーフローの場合、*errno* に ERANGEの値を設定し、戻り値に LONG_MAX(負の場合は LONG_MIN) を返します。

3.9.4 strtod – 文字列を double 型の浮動小数点数値に変換

書式

```
#include <stdlib.h>
double strtod(const char *nptr, char **endptr);
const char *nptr; /* 変換対象文字列へのポインタ */
char **endptr; /* 読み込みの終了位置 */
```

戻り値

変換された数値

解説

strtod関数は文字列を double型の数値に変換します。

入力文字列は、変換する型の数値として解釈できる文字の並びであり、数字の一部として解釈できない文字があると、そこで文字列の入力を中止します。endptrの指す領域には、入力文字列のうち数値として解釈できない最初の文字へのポインタを設定します。

endptrが NULL の場合、この設定は行われません。

変換結果がオーバーフローあるいはアンダーフローの場合、*errno* に ERANGEの値を設定し、戻り値に HUGE_VAL (負の場合は-HUGE_VAL)、あるいは 0 を返します。

3.9.5 strtol – 文字列を long 型の整数値に変換

書式

```
#include <stdlib.h>
long strtol(const char *nptr, char **endptr, int base);
const char *nptr; /* 変換対象文字列へのポインタ */
char **endptr; /* 読み込みの終了位置 */
int base; /* 変換の基数 ( 0,2〜36) */
```

戻り値

‘変換された値’
　　変換された

‘0’　　変換できなかった

解説

strtol関数は、*ntpr* で示す文字列を読み込み、*base* で指定した基数の数値(long 型）に変換します。

strtol関数は、変換対象文字列を先頭から順に読み込み、基数 *base* の数値を表す数字として解釈できない文字か null 文字を検出した時点で読み込みを中止します。

その時指している文字へのポインタを *endptr* に設定します(*endptr* が NULL でない場合のみ)。

なお、変換対象文字列の始めの空白文字(先頭から空白文字以外を検出するまでの文字）は無視され、変換の対象になりません。

base が 0 の場合、*nptr* の指す文字列の先頭の文字から基数を判断します。*base* の値が 2〜36 の間の場合、*base* 値が変換するときの基数となります。

変換対象文字列中の a から z(および A から Z）の文字はそれぞれ 10 から 35 の値に対応付けられ、*base* の値以上の文字は解釈できない文字と認識されます。符号の後の 0、および、*base* が 16 のときの 0x(0X) は無視されます。

変換できなかった場合、戻り値として 0 を返し、*endptr* に *nptr* がセットされます(*endptr* が NULL でない場合のみ)。

変換後の値がオーバーフローを起こす(long で表現できない）場合は、戻り値に LONG_MAX (負の場合は LONG_MIN）を返し、*errno* に ERANGEが設定されます。

3.9.6 strtoul – 文字列を unsigned long 型の整数値に変換

書式

```
#include <stdlib.h>
unsigned long int strtoul(const char *nptr, char **endptr, int base);
const char *nptr; /* 変換対象文字列へのポインタ */
char **endptr; /* 読み込みの終了位置 */
int base; /* 変換の基数 ( 0,2〜36) */
```

戻り値

‘変換された値’
　　変換された

‘0’　　変換できなかった

‘ULONG_MAX’
　　オーバーフロー

解説

strtoul関数は、*nptr* で示す文字列を *base* で指定した基数の数値(unsigned long 型）に変換します。

strtoul関数は、変換対象文字列を先頭から順に読み込み、基数 *base* の数値を表す数字として解釈できない文字か null 文字を検出した時点で読み込みを中止します。

その時指している文字へのポインタを *endptr* に設定します(*endptr* が NULL でない場合のみ)。

なお、変換対象文字列の始めの空白文字(先頭から空白文字以外を検出するまでの文字) は無視され、変換の対象になりません。

base が 0 の場合、*nptr* の指す文字列の先頭の文字から基数を判断します。*base* の値が 2〜36 の間の場合、*base* 値が変換するときの基数となります。

変換対象文字列中の a から z(および A から Z) の文字はそれぞれ 10 から 35 の値に対応付けられ、*base* の値以上の文字は解釈できない文字と認識されます。符号の後の 0、および、*base* が 16 のときの 0x(0X) は無視されます。

変換できなかった場合、戻り値として 0 を返し、*endptr* に *nptr* がセットされます(*endptr* が NULL でない場合のみ)。変換後の値がオーバーフローを起こす(unsigned long で表現できない) 場合は、戻り値に ULONG_MAXを返し、*errno* に ERANGEが設定されます。

3.9.7 rand – 0 から RAND_MAX の間の擬似乱数整数を生成

書式

```
#include <stdlib.h>
int rand(void);
```

戻り値

生成した擬似乱数整数

解説

rand関数は、0 から RAND_MAX の間の擬似乱数整数を生成します。

3.9.8 srand – 擬似乱数数列の初期値を設定

書式

```
#include <stdlib.h>
void srand(unsigned int seed);
unsigned int seed; /* 擬似乱数列生成の初期値 */
```

戻り値

なし

解説

srand関数は、rand関数が擬似乱数列を生成するための初期値を設定します。したがって、rand関数で擬似乱数値を生成しているときに、再度 srand関数で同じ値の初期値を設定すると、擬似乱数列は繰り返し生成されることになります。srand関数よりも rand関数が先に呼ばれた場合、擬似乱数列生成の初期値として 1 が設定されます。

3.9.9 calloc – 記憶域を割り当て、割り当てた記憶域を 0 で初期化

書式

```
#include <stdlib.h>
void  *calloc(size_t nelem, size_t elsize);
size_t nelem; /* 要素の数 */
size_t elsize; /* ひとつの要素の占めるバイト数 */
```

戻り値

'割り当てられた記憶域の先頭のアドレス'
　　正常終了

'NULL'　　エラー(記憶域の割り当てができなかった。引数のいずれかが 0)

解説
calloc関数は、*elsize* バイト単位の記憶域を *nelem* 個割り当てます。割り当てられた記憶域をすべて 0 によって初期化します。

3.9.10 free – 指定された記憶域を開放

書式

```
#include <stdlib.h>
void free(void *ptr);
void *ptr; /* 解放する記憶域の先頭アドレス */
```

戻り値
なし

解説
free関数は、*ptr* が指す記憶域を解放し、再度割り当てて使用できるようにします。*ptr* が NULL であるとき、何もしません。free関数で解放しようとした記憶域、または realloc関数の *ptr* が指す記憶域が、calloc、malloc、realloc関数で割り当てられた記憶域でないとき、または、既に free、realloc関数によって解放されていたときの動作は保証されません。

3.9.11 malloc – 記憶域の割り当て

書式

```
#include <stdlib.h>
void *malloc(size_t size);
size_t size; /* 割り当てる記憶域のバイト数 */
```

戻り値

'割り当てられた記憶域の先頭のアドレス'
　　正常終了

'NULL'　　エラー(記憶域の割り当てができなかった。引数のいずれかが 0)

解説
malloc関数は、*size* で示されるバイト分の記憶域を割り当てます。

3.9.12 realloc – 記憶域の大きさを指定された大きさに変更

書式

```
#include <stdlib.h>
void *realloc(void *ptr, size_t size);
void *ptr; /* 変更する記憶域の先頭アドレス */
size_t size; /* 変更後の記憶域のバイト数 */
```

戻り値

'割り当てられた記憶域の先頭のアドレス'
　　記憶域の割り当てができた

'NULL'　　エラー(記憶域の割り当てができなかった。または引数のいずれかが 0。)

解説

realloc関数は、*ptr* の指す記憶域の大きさを *size* で示されるバイト分の大きさに変更します。

変更後の記憶域が変更前より小さい場合、変更後の記憶域の大きさまでの内容は変化しません。

realloc関数の *ptr* が指す記憶域が、calloc、malloc、realloc関数で割り当てられた記憶域でないとき、または、既に free、realloc関数によって解放されていたときの動作は保証されません。

ptr が NULL の場合、malloc関数のような処理ができます。また、*ptr* が NULL でなく *size* が 0 の場合、記憶域が解放されます。

3.9.13 abort – プログラムの実行を強制終了

書式

```
#include <stdlib.h>
void abort(void);
```

戻り値

なし

解説

abort関数は、プログラムの実行を強制終了します。また、その際にシグナル SIGABRTを発生します。abort関数は、バッファのフラッシュ、ファイルのクローズおよびテンポラリファイルの削除はしません。

3.9.14 atexit – プログラム正常終了時に呼び出される関数を登録

書式

```
#include < stdlib.h>
int atexit(void (*func)(void));
void (*func)(void); /* 登録する関数へのポインタ */
```

戻り値

'0'　　登録できた

'0 以外'　　登録できなかった

解説

atexit関数は、プログラムが正常終了時あるいは exit関数を呼び出した時において呼び出される関数を登録します。関数の最大登録数は 32 個です。プログラムが正常終了した時に実行される関数の順序は登録順序の逆順です。

3.9.15 exit – プログラムを終了

書式

```
#include <stdlib.h>
void exit(int status);
int status; /* 終了ステータス */
```

戻り値

なし

解説

exit関数は、プログラムを終了します。その際の後処理として以下の処理を実行します。

1. オープンされているストリームのバッファをすべてフラッシュします。
2. オープンされているファイルをすべてクローズします。
3. tmpfileで生成したファイルをすべて削除します。
4. atexit関数で登録された関数を登録したときの逆順ですべて実行します。
5. setjmp関数で保存しておいた環境へ、制御が復帰します(本関数を呼び出した側には制御は戻りません)。復帰先の環境において statusの値を知るには_get_exit_code関数を使用します。

3.9.16 getenv – 環境変数の内容を取得

書式

```
#include <stdlib.h>
char *getenv(const char *name);
const char *name; /* 環境変数名 */
```

戻り値

'変数の定義する文字列へのポインタ'
　　name で指定した変数が見つかった

'NULL'　　*name* で指定した変数が見つからなかった

解説

getenv関数は、環境リストを検索し *name* で指定された変数を探します。

注意

getenv関数を使用するためには、getenv関数自身を作成する必要があります。

3.9.17 system – コマンド文字列をホスト環境に渡す

書式

```
#include <stdlib.h>
int system(const char *string);
const char *string; /* コマンド文字列 */
```

戻り値

'0 以外'　　*string* が NULL で、コマンドプロセッサが存在する

'0'　　*string* が NULL で、コマンドプロセッサが存在しない

※ *string* が NULL でないときの戻り値は、処理系に依存します。

解説

system関数は、*string* で指定された文字列をコマンドプロセッサによって実行されるホスト環境に渡します。また、*string* に NULL を指定した場合は、コマンドプロセッサが存在するかどうかを調べます。

注意

system関数を使用するには、system関数自身を作成する必要があります。

3.9.18 bsearch – ２分割検索を行う

書式

```
#include <stdlib.h>
void *bsearch(const void *key, const void *base,
              size_t nmemb, size_t size,
              int (*compar)(const void *p1, const void *p2));
const void key; /* 検索するデータへのポインタ */
const void base; /* 検索対象となるテーブルへのポインタ */
size_t nmemb; /* 検索対象のメンバの数 */
size_t size; /* 検索対象のメンバのバイト数 */
int (*compar)(const void *p1, const void *p2);
/* 比較を行う関数へのポインタ */
```

戻り値

'一致したメンバへのポインタ'
　　一致するメンバが検索できた

'NULL'　　検索できなかった

解説

bsearch関数は、*key* の指すデータと一致するメンバを、*base* の指すテーブルの中で二分検索します。比較を行う関数は、比較する 2 つのデータへのポインタ *p1*(第 1 引数)、*p2*(第 2 引数) を受け取り、次の仕様に従って結果を返してください。

'**p1* < **p2* のとき'
　　負の値を返す

‘*p1* == *p2* のとき’
0 の値を返す

‘*p1* > *p2* のとき’
正の値を返す

なお、検索対象となる各メンバは、昇順に並んでいる必要があります。

3.9.19 qsort – ソートを行う

書式

```
#include <stdlib.h>
void qsort(const void *base, size_t nmemb, size_t size,
           int(*compar)(const void *, const void *));
const void *base; /* ソート対象となるテーブルへのポインタ */
size_t nmemb; /* ソート対象のメンバの数 */
size_t size; /* ソート対象のメンバのバイト数 */
int (*compar)(const void *, const void *);
/* 比較を行う関数へのポインタ */
```

戻り値

なし

解説

qsort関数は、*base* の指すテーブルのデータをソートします。データの並ぶ順序は比較関数 *compar()* に依存します(仕様は bsearch関数における比較関数 *compar()* と同じ)。

3.9.20 abs – int 型整数の絶対値を計算

書式

```
#include <stdlib.h>
int abs(int j);
int j; /* 絶対値を求める整数 */
```

戻り値

演算結果の絶対値

解説

abs関数は int型の絶対値を計算します。演算結果が int型で表現できないときの動作は保証されません。

3.9.21 div – int 型整数の除算の商と余りを計算

書式

```
#include <stdlib.h>
div_t div(int number, int denom);
int number; /* 被除数 */
int denom; /* 除数 */
```

戻り値

演算結果の商と余り

解説

div関数は、*number* を *denom* で除算した結果の商と余りを計算します。戻り値は div_t構造体で、そのメンバの quotに商、remに余りが入ります。

3.9.22 labs – long 型整数の絶対値を計算

書式

```
#include <stdlib.h>
long labs(long j);
long j; /* 絶対値を求める整数 */
```

戻り値

演算結果の絶対値

解説

labs関数は long型整数の絶対値を計算します。演算結果が long型で表現できないときの動作は保証されません。

3.9.23 ldiv – long 型整数の除算の商と余りを計算

書式

```
#include <stdlib.h>
ldiv_t ldiv(long number, long denom);
long number; /* 被除数 */
long denom; /* 除数 */
```

戻り値

演算結果の結果の商と余り

解説

ldiv関数は、*number* を *denom* で除算した結果の商と余りを計算します。戻り値は ldiv_t 構造体で、そのメンバ quot に商、rem に余りが入ります。

3.9.24 mblen – マルチバイト文字のバイト数を算出

書式

```
#include <stdlib.h>
int mblen(const char *s, size_t n);
const char *s; /* マルチバイト文字へのポインタ */
size_t n; /* 検査バイト数 */
```

戻り値

‘< *s* が NULL の場合>’

‘0 以外’　マルチバイト文字セットが状態に依存する。

‘0’　マルチバイト文字セットが状態に依存しない。

‘< *s* が NULL でない場合>’

‘バイト数’　*s* が指すマルチバイト文字列のバイト数。

‘0’　　s が null 文字を指している。

‘-1’　　s が指す文字列がマルチバイト文字列でない。

解説

mblen関数は、s が示すマルチバイト文字列のバイト数を返します。なお、mbtowc関数が、シフト状態に影響を受けなければ mblenは以下の例と同値です。

例:　mbtowc((wchar_t *)0, s, n);

注意

現在、マルチバイト文字に対する最大値は 1 バイトだけをサポートしています。ロケールは"C"環境のみをサポートしています。

3.9.25 mbtowc – マルチバイト文字をワイド文字に変換

書式

```
#include <stdlib.h>
int mbtowc(wchar_t *pwc, const char *s, size_t n);
wchar_t *pwc; /* ワイド文字の格納バッファ */
const char *s; /* マルチバイト文字へのポインタ */
size_t n; /* 検査バイト数 */
```

戻り値

‘変換マルチバイト文字数’
　　pwc、s が共に NULL でない

‘0’　　s が null 文字を指している

‘-1’　　s がマルチバイト文字を正しく指していない

※ s が NULL のとき、または、pwc と s が共に NULL のときは、その時点でのマルチバイト文字セットが状態に依存すれば 0 以外を返し、そうでなければ 0 を返します。

解説

mbtowc関数は、s が示すマルチバイト文字に対して以下のような処理を行います。

‘pwc および s が NULL 以外のとき’
　　s が指し示すところから n バイト分のマルチバイト文字をワイド文字に変換し、pwc の指すバッファに格納します。

‘pwc が NULL で s が NULL 以外のとき’
　　mblen関数と同じ動作をします。

‘s が NULL、または pwc と s が共に NULL のとき’
　　その時点でのマルチバイト文字セットが状態に依存するかどうかをチェックします。

注意

マルチバイト文字に対する最大値は 1 バイトだけをサポートしています。ロケールは"C"環境のみをサポートしています。

3.9.26 wctomb – ワイド文字をマルチバイト文字に変換

書式

```
#include <stdlib.h>
int wctomb(char *s, wchar_t wchar);
char *s; /* マルチバイト文字格納バッファ */
wchar_t wchar;  /* ワイド文字 */
```

戻り値

＜ *s* が NULL の場合＞

'0'　その時点でのマルチバイト文字セットが状態に依存しない。

'0 以外'　その時点でのマルチバイト文字セットが状態に依存する。

＜ *s* が NULL でない場合＞

'バイト数'　変換されたマルチバイト文字のバイト数

'0'　*wchar* が 0

'-1'　*wchar* に対応するマルチバイト文字が存在しない

解説

マルチバイト文字に対する最大値は 1 バイトだけをサポートしています。ロケールは"C"環境のみをサポートしています。wctomb 関数は、*wchar* で示されるワイド文字をそれに対応するマルチバイト文字に変換して結果を *s* が指し示すバッファに格納します。*s* が NULL の場合、その時点でのマルチバイト文字セットが状態に依存するかどうかをチェックします。

3.9.27 mbstowcs – マルチバイト文字列をワイド文字列に変換

書式

```
#include <stdlib.h>
size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
wchar_t *pwcs; /* ワイド文字列の格納バッファ */
const char *s; /* マルチバイト文字列へのポインタ */
size_t n; /* 変換するワイド文字数 */
```

戻り値

'変換した文字数'
　正常終了(ただし、終端 null 文字は文字数に含まない)

'-1'　変換中にマルチバイト文字でない文字を検出したときは-1 を size_t にキャストして返します

解説

mbstowcs関数は、*s* で示されるマルチバイト文字列をそれに対応するワイド文字の文字列に n で示される文字数分変換し、結果を *pwcs* の示すバッファに返します。

注意

マルチバイト文字に対する最大値は 1 バイトだけをサポートしています。ロケールは"C"環境のみをサポートしています。

3.9.28 wcstombs – ワイド文字列をマルチバイト文字列に変換

書式

```
#include <stdlib.h>
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);
char *s; /* マルチバイト文字列格納バッファ */
const wchar_t *pwcs; /* ワイド文字列 */
size_t n;  /* 書き込みバイト数 */
```

戻り値

'変換した文字数'
: 正常に変換された(ただし、終端 null 文字は文字数に含みません)

'-1'
: 変換中に正しくないワイド文字が検出された(-1 を size_t にキャストして返します)

解説

マルチバイト文字に対する最大値は 1 バイトだけをサポートしています。ロケールは"C"環境のみをサポートしています。wcstombs関数は、*pwcs* で示されるワイド文字列を *n* で示されるバイト数分マルチバイト文字に変換して、その結果を *s* の指し示すバッファに格納します。

3.10 string.h – libc.a

3.10.1 memcpy – 複写元記憶域の内容を指定分、複写先記憶域に複写

書式

```
#include <string.h>
void *memcpy(void *s1, const void *s2, size_t n);
void *s1; /* 複写先 */
const void *s2; /* 複写元 */
size_t n; /* 複写するバイト数 */
```

戻り値

複写先の記憶域へのポインタ

解説

memcpy関数は、複写元の記憶域の内容を指定された大きさ分、記憶域に複写します。

3.10.2 memmove – 複写元記憶域の内容を指定分、複写先記憶域に移動

書式

```
#include < string.h>
void *memmove(void *s1, const void *s2, size_t n);
void *s1; /* 複写先( 格納バッファ ) */
const void *s2; /* 複写元 */
size_t n; /* 複写するバイト数 */
```

戻り値

s1

解説

memmove関数は、*s2* の内容の先頭から *n* バイト分 *s1* に複写します。memmove関数は、*s1* と *s2* のオブジェクト(メモリ領域) が重複しても複写は正常に行われます(ただし、重複部分の、複写元のデータは失われます)。複写した結果、*s1* のバッファ領域を超えるような場合の動作は保証されません。

3.10.3 strcpy – 複写元文字列を、複写先記憶域に null 文字も含めて複写

書式

```
#include <string.h>
char *strcpy(char *s1, const char *s2);
char *s1; /* 複写先の記憶域へのポインタ */
const char *s2; /* 複写元の文字列へのポインタ */
```

戻り値

複写先の記憶域へのポインタ

解説

strcpy関数は、複写元の文字列を指定された記憶域に null 文字も含めて複写します。

3.10.4 strncpy – 複写元文字列を n 文字分、複写先記憶域に複写

書式

```
#include <string.h>
char *strncpy(char *s1, const char *s2,  size_t n);
char *s1; /* 複写先の記憶域へのポインタ */
const char *s2; /* 複写元の文字列へのポインタ */
size_t n; /* 複写する文字数 */
```

戻り値

複写先の記憶域へのポインタ

解説

strncpy関数は、複写元の文字列 *s2* を指定された文字数分、指定された記憶域 *s1* に複写します。*s2* で指定された文字列の長さが *n* 文字より短い場合、*n* 文字になるまで null 文字が付加されます。逆に文字列 *s2* が *n* 文字以上の場合、*s1* に複写された文字列は null 文字で終了しないことになります。

3.10.5 strcat – 文字列の後に、文字列を連結

書式

```
#include <string.h>
char *strcat(char *s1, const char *s2);
char *s1; /* 連結される文字列へのポインタ */
char *s2; /* 連結する文字列へのポインタ */
```

戻り値

連結後の文字列へのポインタ *s1*

解説

strcat関数は、文字列 *s1* の後に文字列 *s2* を連結します。このとき、文字列 *s1* の最後の null 文字は *s2* の先頭文字で置き換えられ、連結後の文字列 *s2* の最後には必ず null 文字が付きます。なお strncat関数では、連結する文字数を引数 *n* で指定できます。

3.10.6 strncat – 文字列を指定した文字数分連結

書式

```
#include <string.h>
char *strncat(char *s1,  const char *s2,  size_t n);
char *s1; /* 連結される文字列へのポインタ */
const char *s2; /* 連結する文字列へのポインタ */
size_t n; /* 連結する文字数 */
```

戻り値

連結後の文字列へのポインタ *s1*

解説

strncat関数は、文字列 *s1* の後に文字列 *s2* を連結します。このとき、文字列 *s1* の最後の null 文字は *s2* の先頭文字で置き換えられ、連結後の文字列 *s2* の最後には必ず null 文字が付きます。連結する文字数は引数 *n* で指定できます。

3.10.7 memcmp – 二つの記憶域を比較

書式

```
#include <string.h>
int memcmp(const void *s1, const void *s2, size_t n);
const void *s1; /* 比較される記憶域へのポインタ */
const void *s2; /* 比較する記憶域へのポインタ */
size_t n; /* 比較する記憶域の文字数 */
```

戻り値

‘正の値’　*s1* の内容 > *s2* の内容

‘0’　*s1* の内容 == *s2* の内容

‘負の値’　*s1* の内容 < *s2* の内容

解説

memcmp関数は、*s1* で指定された記憶域と *s2* で指定された記憶域の最初の *n* 文字分の内容を比較します。

3.10.8 strcmp – 二つの文字列を比較

書式

```
#include <string.h>
int strcmp(const void *s1, const void *s2);
const void *s1; /* 比較される文字列へのポインタ */
```

```
const void *s2; /* 比較する文字列へのポインタ */
```

戻り値

‘正の値’　*s1* の内容 > *s2* の内容

‘0’　*s1* の内容 == *s2* の内容

‘負の値’　*s1* の内容 < *s2* の内容

解説

strcmp関数は、*s1* で指定された文字列と、*s2* で指定された文字列の内容を比較し、その結果を戻り値として設定します。

3.10.9 strcoll – ロケールに基づいて２つの文字列を比較

書式

```
#include <string.h>
int strcoll(const char *sl, const char *s2);
const char *s1; /* 比較文字列１ */
const char *s2; /* 比較文字列２ */
```

戻り値

‘正の値’　*s1* の内容 > *s2* の内容

‘0’　*s1* の内容 == *s2* の内容

‘負の値’　*s1* の内容 < *s2* の内容

解説

ロケールは"C"環境のみサポートしています。strcoll関数は、現在のロケールが示すLC_COLLATE（ locale.h に定義）に基づいて（ 変換して）*s1* と *s2* を比較します。

3.10.10 strncmp – 二つの文字列を指定文字数分比較

書式

```
#include <string.h>
int strncmp(const void *s1, const void *s2, size_t n);
const void *s1; /* 比較される文字列へのポインタ */
const void *s2; /* 比較する文字列へのポインタ */
size_t n; /* 比較する文字数の最大値 */
```

戻り値

‘正の値’　*s1* の内容 > *s2* の内容

‘0’　*s1* の内容 == *s2* の内容

‘負の値’　*s1* の内容 < *s2* の内容

解説

strncmp関数は、*s1* で指定された文字列と、*s2* で指定された文字列の内容を、引数 *n* で指定された文字数分比較し、その結果を戻り値として設定します。

3.10.11 strxfrm – ロケールに基づいて文字列を変換

書式

```
#include <string.h>
size_t strxfrm(char *s1, const char *s2, size_t n);
char *s1; /* 変換結果文字列格納バッファ */
const char *s2; /* 変換文字列 */
size_t n; /* 変換文字数 */
```

戻り値

変換後の文字列のバイト数(文字列の最後の null 文字は含みません)。戻り値が n より大きいとき *s1* の内容は不定です。

解説

ロケールは"C"環境のみサポートしています。*strxfrm* 関数は、現在のロケールが示すLC_COLLATEに基づいて *s2* の文字列を最初の文字から n バイト分変換し *s1* に変換結果を格納します。

3.10.12 memchr – 記憶域中、指定された文字が最初に現われる位置を検索

書式

```
#include <string.h>
void *memchr(const void *s, int c, size_t n);
const void *s; /* 検索を行う記憶域へのポインタ */
int c; /* 検索する文字 */
size_t n; /* 検索を行う文字数 */
```

戻り値

'文字位置へのポインタ'
　　検索の結果文字が見つかった

'NULL'　　検索の結果文字が見つからなかった

解説

memchr関数は、指定された記憶域の先頭から n 文字の中で最初に現れた文字 c と同一文字の位置へのポインタを戻り値として返します。

3.10.13 strchr – 指定文字が最初に現われる位置を検索

書式

```
#include <string.h>
char *strchr(const char *s, int c);
const char *s; /* 検索を行う文字列へのポインタ */
int c; /* 検索する文字 */
```

戻り値

'検索の結果見つかった文字へのポインタ'
　　文字が見つかった

'NULL'　　文字が見つからなかった

解説

strchr関数は、文字列 s 中から文字 c を検索し、それが最初に現れた位置へのポインタを戻り値として返します。

3.10.14 strcspn – 文字列中、指定外文字が何文字続くか算出

書式

```
#include <string.h>
size_t strcspn(const char *s1, const char *s2);
const char *s1; /* 調べられる文字列へのポインタ */
const char *s2; /* 指定文字から構成される文字列へのポインタ */
```

戻り値

検索結果の文字列長

解説

strcspn関数は、文字列 $s1$ の先頭から、文字列 $s2$ を構成する文字のいずれも出現しない部分が何文字分続くかを調べ、その長さを戻り値として返します。

3.10.15 strpbrk – 指定文字が最初に現われる位置を検索

書式

```
#include <string.h>
char *strpbrk(const char *s1, const char *s2);
const char *s1; /* 検索する領域( 文字列) へのポインタ */
const char *s2; /* 探したい文字セット( 文字列) へのポインタ */
```

戻り値

'検索の結果見つかった文字へのポインタ'
　　文字が見つかった

'NULL'　　文字が見つからなかった

解説

strpbrk関数は、文字列 $s1$ 内の先頭から、文字列 $s2$ を構成する文字のいずれかが最初に現れる位置を検索し、その位置へのポインタを戻り値として返します。

3.10.16 strrchr – 指定文字が最後に現われる位置を検索

書式

```
#include <string.h>
char *strrchr(const char *s, int c);
const char *s; /* 検索を行う文字列へのポインタ */
int c; /* 検索する文字 */
```

戻り値

‘検索の結果見つかった文字へのポインタ’
　　文字が見つかった

‘NULL’　　文字が見つからなかった

解説

strrchr関数は、文字列 *s* 中から文字 *c* を検索し、最後に現れた位置を求め、そのポインタを戻り値として返します。文字列 *s* の終端を示す null 文字も検索の対象として含みます。

3.10.17 strspn – 指定文字が先頭から何文字続くかを算出

書式

```
#include <string.h>
size_t strspn(const char *s1, const char *s2);
const char *s1; /* 検索する領域( 文字列) へのポインタ */
const char *s2; /* 指定文字から構成される文字列へのポインタ */
```

戻り値

検索結果の文字列長

解説

strspn関数は、文字列 *s1* の先頭から、文字列 *s2* を構成する文字のいずれかと一致する部分が何文字分続くかを調べ、その長さを戻り値として返します。

3.10.18 strstr – 指定文字列が最初に現われる位置を検索

書式

```
#include <string.h>
char *strstr(const char *s1, const char *s2);
const char *s1; /* 検索を行う文字列へのポインタ */
const char *s2; /* 検索する文字列へのポインタ */
```

戻り値

‘検索の結果見つかった文字へのポインタ’
　　文字が見つかった

‘NULL’　　文字が見つからなかった

解説

strstr関数は、文字列 *s1* において、文字列 *s2* が最初に現れる位置を検索し、その位置へのポインタを戻り値として返します。*s2* が NULL の場合、戻り値は NULL になります。

3.10.19 strtok – 文字列をトークン(字句) に分割

書式

```
#include <string.h>
char *strtok(char *s1, const char *s2);
char *s1; /* 字句に切り分ける文字列へのポインタ */
const char *s2; /* 文字列を切り分けるための文字列へのポインタ */
```

戻り値

'切り分けた字句の先頭へのポインタ'
　　字句の切り分けを行った

'NULL'　　字句が切り分けられなかった

解説

strtok関数は、文字列 *s2* 中の文字を区切り子として、文字列 *s1* をいくつかのトークン(字句) に切り分けます。strtok関数は連続的に呼び出され、最初の呼び出し時は文字列 *s1* の先頭から、2 回目以降は以前に切り分けられた字句の次の文字から、文字列 *s2* 中の文字によって字句に切り分けます。2 回目以降の呼び出し時は、第 1 引数に NULL を指定します。第 2 引数 *s2* の内容は呼び出しの度に変わってもかまいません。切り出された字句の最後には null 文字が付きます。

3.10.20 memset – 記憶域の先頭から指定された文字を指定文字数分設定

書式

```
#include <string.h>
void *memset(void *s, int c, size_t n);
void *s; /* 文字が設定される記憶域へのポインタ */
int c; /* 設定する文字 */
size_t n; /* 設定する文字数 */
```

戻り値

文字を設定した記憶域へのポインタ s

解説

memset関数は、記憶域 s に n 文字分、文字 c を設定します。

3.10.21 strerror – エラーメッセージを取得

書式

```
#include <string.h>
char *strerror(int e);
int e; /* エラー番号 */
```

戻り値

エラー番号 e に対応するエラーメッセージへのポインタ

解説

strerror関数は、エラー番号 e に対応するエラーメッセージへのポインタを戻り値として返します。

3.10.22 strlen – 文字列の長さを計算

書式

```
#include <string.h>
size_t strlen(const char *s);
const char *s; /* 長さを求める文字列へのポインタ */
```

戻り値

計算した文字列の文字数

解説

strlen関数は、文字列 *s* の長さを計算します。ただし、文字列の終端を示す null 文字は数えません。

3.11 time.h – libc.a

3.11.1 clock – プログラムの実行時間の算出

書式

```
#include <time.h>
clock_t clock(void);
```

戻り値

未サポート

解説

clock関数は、使用するシステムの環境に依存します。そのため clock関数はサポートされていません。

注意

clock関数を使用するためには、clock関数自身を作成する必要があります。

3.11.2 difftime – 指定された２つの時間差を算出

書式

```
#include <time.h>
double difftime(time_t time1, time_t time2);
time_t time1; /* 時間１ */
time_t time2; /* 時間２ */
```

戻り値

time1 から *time2* を引いた値(秒単位)

解説

difftime関数は、*time1* から *time2* を引いた値を求めます。

3.11.3 mktime – tm 型構造体で示される日時を暦時間に変換

書式

```
#include <time.h>
time_t mktime(struct tm *timeptr);
struct tm *timeptr; /* tm 型構造体へのポインタ */
```

戻り値

‘変換結果’　正常に暦時間に変換終了

'-1'　　変換ができなかった(-1 を time_t 型にキャストして返します)

解説

mktime関数は、*timeptr* で示される tm 型構造体の要素別の時間(broken-down time)を暦時間に変換します。なお、mktime関数では *timeptr* で指定された tm 型構造体を以下のように取り扱います。

- tm 型構造体の tm_wday と tm_yday のオリジナルの値(読み込んだときの値)は無視します。
- *timeptr* の tm 型構造体の内容は、暦時間に合うように更新されます。

3.11.4 time – 現在の暦時間を求める

書式

```
#include <time.h>
time_t time(time_t *timer);
time_t *timer; /* 暦時間 */
```

戻り値

'現在の暦時間'
　　timer が NULL でない

'-1'　　現在の暦時間が得られないとき(-1 を time_t 型にキャストして返します)

解説

time関数は、現在の暦時間を *timer* の指し示すバッファに結果を書き込みます。

注意

time関数を使用するには、time関数自身を作成する必要があります。

3.11.5 asctime – tm 型構造体の日時をテキスト文字列形式に変換

書式

```
#include < time.h>
char *asctime(const struct tm *timeptr);
const struct tm timeptr; /* tm 型構造体へのポインタ */
```

戻り値

変換した文字列へのポインタ

解説

sctime関数は、*timeptr* で示された日時を以下の例のような形式に変換します。

```
Thu May 12 16:00:00 1995\n\0
```

3.11.6 ctime – time_t 型の暦時間をテキスト文字列の形式に変換

書式

```
#include <time.h>
char *ctime(const time_t *timer);
const time_t *timer; /* 暦時間 (calendar time) */
```

戻り値

変換した文字列へのポインタ

解説

ctime関数は、*timer* で示される暦時間を以下の例のような形式に変換します。

```
Thu May 12 16:00:00 1995\n\0
```

3.11.7 gmtime – 暦時間を世界標準時に変換

書式

```
#include <time.h>
struct tm *gmtime(const time_t *timer);
const time_t *timer; /* 時間 */
```

戻り値

'変換した結果(tm 型構造体へのポインタ) '
　　正常終了

'NULL'　　*timer* を世界標準時に変換できなかった

解説

gmtime関数は、*timer* で示す暦時間を世界標準時(UTC) に変換し、tm 型 (broken-down time) に変換します。

3.11.8 localtime – 暦時間を現在の現地時間に変換

書式

```
#include <time.h>
struct tm *localtime(const time_t *timer);
const time_t *timer; /* 時間 */
```

戻り値

変換した結果(tm 型構造体へのポインタ)

解説

localtime関数は、*timer* で示す暦時間を現地時間(local time) に変換して、tm 型に変換します。

注意

ロケールは"C"環境のみサポートしています。localtime関数を使用するためには、この関数が(関数内部で) 呼び出している getenv関数を作成する必要があります。

3.11.9 strftime – tm 型構造体で示される日時を書式変換

書式

```
#include <time.h>
size_t strftime(char *s, size_t maxsize,
                const char *format, const struct tm *timeptr);
char *s; /* 変換文字列格納バッファ */
size_t maxsize;  /* 変換文字数の最大値 */
const char *format;       /* 書式文字列 */
const struct tm *timeptr; /* tm 型構造体へのポインタ */
```

戻り値

'書き込んだ文字数'
: *s* に書き込んだ文字数が *maxsize* 以下

'0'
: 書き込んだ文字数が *maxsize* を超えた(このとき *s* の内容は不定)

解説

ロケールは"C"環境のみサポートしています。strftime関数は、*timeptr* で示される tm 型構造体の broken-down time を *format* で指定された書式に従って変換し *s* に変換結果を格納します。そのときの変換結果は、最大 *maxsize* 数まで *s* に格納されます。以下に *format* 中で使用できる各変換指定子を示します。これら以外の変換指定子を指定した場合の動作は不定です。

'変換指定子'
: 意味

'%a'
: ロケールでの曜日の略語

'%A'
: ロケールでの曜日の名前

'%b'
: ロケールでの月の略語

'%B'
: ロケールでの月の名前

'%c'
: ロケールでの適切な日付と時刻の表現

'%d'
: 月の中での日付を整数で表したもの(01 から 31)

'%H'
: 時間(24 時間制)を整数で表したもの(00 から 23)

'%I'
: 時間(12 時間制)を整数で表したもの(01 から 12)

'%j'
: 年初からの日数を整数で表したもの(001 から 366)

'%m'
: 月を整数で表したもの(01 から 12)

'%M'
: 分を整数で表したもの(00 から 59)

'%p'
: 12 時間を表すそのときのロケールの午前 (AM)と午後(PM)

'%S'
: 秒を整数で表したもの(00 から 59)

'%U'
: 年初からの週数を整数で表したもの(ただし、日曜を週の最初の日とします (00 から 53))

'%w'
: 日曜を 0(最初の日)としたときの週の中での日を整数で表したもの(0 から 6)

'%W'
: 年初からの週数を整数で表したもの(ただし、月曜を週の最初の日とします (00 から 53))

‘%x’　　ロケールでの適切な日付の表現

‘%X’　　ロケールでの適切な時刻の表現

‘%y’　　世紀中での年数を整数で表したもの(00 から 99)

‘%Y’　　西暦を整数で表したもの

‘%Z’　　時間帯またはその略語を表したもの(時間帯が特定できないときは NULL 文字で表します)

‘%%’　　%自身を表します

4 非 ANSI 関数

非 ANSI 関数は、C 標準ライブラリを使用するための初期化処理、終了処理を行う関数です。これらの関数は、標準ライブラリ (libc.a) に含まれています。

C 標準ライブラリ用 main 関数の_c_main関数以外、通常は呼び出す必要はありません。

4.1 _action_atexit – ユーザー終了処理関数の実行

書式

```
void _action_atexit(void);
```

戻り値

なし

解説

atexit関数によって登録されたユーザーの終了処理関数を実行します。登録関数をすべて実行終了した後、登録関数のカウンタを初期化します。

注意

本関数は通常、main関数実行後に_c_main関数内で呼び出されます。

4.2 _c_main – C 標準ライブラリ用 main

書式

```
void _c_main(void);
```

戻り値

なし

解説

C 標準ライブラリを使用できるようにするために、ライブラリの初期化を行います。その後、main関数を呼び出します。main関数が終了後、ライブラリの終了処理を行い、_exit 関数を呼び出すことによりプログラムを終了します。

_c_main 関数のソースを以下に示します。

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

extern void _init_mem( void );
extern void _exit_mem( void );
extern void _init_stdio( void );
extern void _exit_stdio( void );
extern void _init_atexit( void );
extern void _action_atexit( void );
extern void _init_base_year( int );
extern int _get_exit_code( void );
```

```
extern jmp_buf _900_init_env;

extern void _exit( int );
extern int main(void);
#define BASICYEAR 1970

void _c_main( void )
{
    int exit_code;

    _init_mem();
    _init_stdio();
    _init_atexit();
    _init_base_year( BASICYEAR );
    srand( 1 );
    errno = 0;
    if ( ! setjmp( _900_init_env ) )
        exit_code = main();
    else
        exit_code = _get_exit_code();

    _action_atexit();
    _exit_stdio();
    _exit_mem();
    _exit( exit_code );
}
```

注意

C 標準ライブラリを使用する場合、スタートアッププログラムから_c_mainを必ず呼び出してください。デフォルトではmain関数にパラメータを渡すことはできません。

4.3 _exit_mem – メモリ操作の終了処理

書式

```
void _exit_mem(void);
```

戻り値

なし

解説

使用されたメモリ領域をすべて解放し、メモリ管理ポインタを初期化します。プログラム中にある入出力関数、標準処理関数、および、ロケール操作関数がすべて終了した後に 1 回だけ呼び出します。

注意

本関数は通常、main関数終了後に_c_main関数内で呼び出されます。

4.4 _exit_stdio – 入出力関数の終了処理

書式

```
void _exit_stdio(void);
```

戻り値

なし

解説

_exit_stdio関数は、すべての入出力ストリームをクローズします。プログラム中にある入出力関数がすべて終了した後に 1 回だけ呼び出します。

注意

本関数は通常、main関数実行後に_c_main関数内で呼び出されます。

4.5 _get_exit_code – 終了コードの取得

書式

```
int _get_exit_code(void);
```

戻り値

exit関数が返した終了コード

解説

_get_exit_code関数は、exit関数が返す終了コードを取り出します。

注意

本関数は通常、exit関数によって終了した関数の終了コードを知るため_c_main関数内で呼び出されます。

4.6 _init_atexit – ユーザー終了処理関数の初期化

書式

```
void _init_atexit(void);
```

戻り値

なし

解説

_init_atexit関数は、atexit関数により登録される登録関数のカウンタを初期化します。初めて atexit関数を使用する前に 1 回だけ呼び出します。

注意

本関数は通常、main関数実行前に_c_main関数内で呼び出されます。main関数終了後、本関数に対応するユーザー終了処理関数の実行関数_action_atexitを呼び出さなければ、ユーザー終了処理関数は実行されません。

参照

Section 4.1 [_action_atexit], page 95

4.7 _init_base_year – 日付、時間操作関数の初期化

書式

```
void _init_base_year(int year);
int year /* 起算年 */
```

戻り値

なし

解説

_init_base_year関数は、起算年を *year* に初期化します。初めて日付・時刻操作関数を使用する前に 1 回だけ呼び出します。

注意

本関数は通常、main関数実行前に_c_main関数内で呼び出されます。また、この時の *year* は 1997 に設定されます。

4.8 _init_mem – メモリ操作関数の初期化

書式

```
void _init_mem(void);
```

戻り値

なし

解説

_init_mem関数は、メモリ管理ポインタを初期化します。初めて入出力関数、標準処理関数、および、ロケール操作関数のいずれかに属する関数を使用する前に 1 回だけ呼び出します。

注意

本関数は通常、main関数実行前に_c_main関数内で呼び出されます。

参照

Section 4.3 [_exit_mem], page 96

4.9 _init_stdio – 入出力処理の初期化

書式

```
void _init_stdio(void);
```

戻り値

なし

解説

_init_stdio関数は、標準入力、標準出力、および標準エラー出力ストリーム用の初期設定および FILE 構造体の初期設定を行います。初めて入出力関数を使用する前に 1 回だけ呼び出します。
本関数を呼び出すことにより、標準入力、標準出力、標準エラー出力はそれぞれ次のモードでオープンされます。また、バッファを使用する場合 1 つの FILE 構造体につき 1036 バイトを使用します。

標準入力　ファイル番号 0、行バッファリング (_IOLBF)

標準出力　ファイル番号 1、行バッファリング (_IOLBF)

標準エラー出力
　　ファイル番号 2、バッファリングなし (_IONBF)

注意

本関数は通常、main関数実行前の_c_main関数内でに呼び出されます。

参照

_exit_stdio

5 低水準ライブラリ

5.1 低水準ライブラリの概要

標準入出力、メモリ管理、シグナル処理、および、時間操作などのライブラリ関数の一部の機能は、ターゲットシステムに依存します。C 標準ライブラリでは、これらのターゲットシステムに依存した機能を低水準関数(低水準ライブラリ) として分離し、各機能を実現する処理関数 (低水準関数) の入出力仕様を定義しています。

C 標準ライブラリでは、ターゲットシステムに依存する機能を、次に示す低水準関数を呼び出すことで実現しています。

低水準関数名	機能
open	ファイルのオープン
close	ファイルのクローズ
read	ファイルからの読み出し
write	ファイルへの書き込み
lseek	ファイルの読み出し／書き込みの位置の設定
_get_core	メモリ領域の確保
_rel_core	メモリ領域の解放
getuniqnum	プロセスごとのユニークな番号を取得
_strerror	エラー番号に対応するエラーメッセージの取得
_exit	プログラムの終了

また、ANSI 規格ではエントリ関数と規定されているものでも、C 標準ライブラリでは低水準関数と位置付けられているものとして次の関数があります。これらの関数の仕様は、「C 標準ライブラリ関数詳細」を参照してください。

ヘッダ	ライブラリ関数名	機能
signal.h	raise	シグナルを送る
	signal	シグナルを受けた場合の処理を指定する
stdio.h	remove	ファイルを削除する

	rename	ファイル名を変更する
stdlib.h	getenv	環境変数を得る
	system	パラメータを実行されるホスト環境に渡す
time.h	clock	実行時間を取り出す
	time	現在の暦時間を得る

したがって、C プログラム内でターゲットシステムに依存する機能を使用する場合、必要な低水準関数をユーザーが作成する必要があります。

これらの低水準関数を、ターゲットシステムに合わせて作成して頂くことにより、C 標準ライブラリを様々なターゲットシステムにて動作させることができます。

5.2 C 標準ライブラリとの対応

C 標準ライブラリ関数のうち、低水準ライブラリを必要とする関数の対応を以下に示します。

分類	C 標準ライブラリ関数	必要な低水準関数
assert.h	assert	_get_core, _rel_core, write, lseek, raise
locale.h	localeconv	_get_core, _rel_core
	setlocale	_get_core, _rel_core
signal.h	raise	raise
	signal	signal
stdio.h	perror	write, _get_core, _rel_core, lseek, _strerror
	fclose	write, getuniqnum, _rel_core, close, remove, lseek
	fflush	write, lseek
	fopen	open, close, lseek
	freopen	write, getuniqnum, _rel_core, close, open, remove, lseek
	remove	remove
	rename	rename
	tmpfile	getuniqnum, open, close, lseek

tmpnam	getuniqnum
fgetpos	lseek
fseek	lseek, write
fsetpos	lseek, write
ftell	lseek
rewind	lseek, write
fgetc	read, _get_core, _rel_core
fgets	read, _get_core, _rel_core
fputc	write, _get_core, lseek, _rel_core
fputs	write, _get_core, lseek, _rel_core
getc	read, _get_core, _rel_core
getchar	read, _get_core, _rel_core
gets	read, _get_core, _rel_core
putc	write, _get_core, _rel_core, lseek
putchar	write, _get_core, _rel_core, lseek
puts	write, _get_core, _rel_core, lseek
ungetc	_get_core, _rel_core
fread	read, _get_core, _rel_core
fwrite	write, _get_core, _rel_core, lseek
fprintf	write, _get_core, _rel_core, lseek
fscanf	read, _get_core, _rel_core
printf	write, _get_core, _rel_core, lseek
scanf	read, _get_core, _rel_core
sprintf	write, _get_core, _rel_core, lseek
sscanf	read, _get_core, _rel_core
setbuf	_rel_core
setvbuf	_rel_core
vfprintf	write, _get_core, _rel_core, lseek
vprintf	write, _get_core, _rel_core, lseek

	vsprintf	write, _get_core, _rel_core, lseek
stdlib.h	abort	raise
	calloc	_get_core, _rel_core
	free	_rel_core
	getenv	getenv
	malloc	_get_core, _rel_core
	realloc	_rel_core, _get_core
	system	system
time.h	asctime	write, _get_core, _rel_core, lseek
	clock	clock
	ctime	write, _get_core, getenv, _rel_core, lseek
	localtime	getenv
	strftime	write, _get_core, getenv, _rel_core, lseek
	time	time

作成した低水準ライブラリを使用するために必要な初期設定は、C プログラム起動前に実行します。必要な低水準ライブラリの初期設定プログラムは、スタートアッププログラム中に記述してください。

5.3 低水準ライブラリによる入出力について

ファイルは、標準ライブラリレベルの標準入出力関数では、FILE 型のデータを使って管理しています。これに対し、低水準ライブラリでは、ファイルに「 ファイル番号」を割り当てて管理します。ファイル番号は、実際のファイルと 1 対 1 に対応する 0 以上の整数です。

ファイル番号 0、1、2 には、それぞれ標準入力、標準出力、標準エラー出力にあらかじめ割り当てておきます。これらは、0 はコンソールからの入力、1 および 2 はコンソールへの出力とするのが一般的です。ファイル番号 0〜2 に対しては、実行環境において矛盾のないように低水準ライブラリを作成してください。とくに、0〜2 がすでに割り当てられているときに open 関数が 0〜2 の値を返さないように注意してください。

低水準ライブラリの open 関数は、与えられたファイルのパス名 (ファイルシステムの中でファイルを識別するための名前) に対してファイル番号を与えます。open 関数では、このファイル番号によってファイルの入出力ができるように、以下の情報の管理が必要な場合があります。

■ デバイスファイルの種類
　　コンソール、プリンタ、ディスクファイル等の特殊なデバイスに対しては、それぞれに適した処理を行う必要があるため、特別なファイル名をシステムで決めておき、open 関数で判定するなど考慮が必要です。

■ バッファの位置、サイズ等の情報
　　ファイルのデータをバッファリングする場合に必要です。

■ ファイル内のバイトオフセット位置
　　ディスクファイルへの入出力などの場合、ファイルの先頭から次に読み出しまたは書き込みを行う位置までのバイトオフセット位置を保持する必要があります。

open 関数で管理した情報にもとづいて、以後、そのファイルに対して入出力 (read/write 関数)、読み出し/書き込み位置の設定 (lseek 関数) を行います。close 関数では、ファイルから読み込んだデータのバッファリングを行っている場合、バッファの内容を読み出して実際のファイルに書き込み、open 関数で設定したデータの領域が再使用できるようにしてください。

5.4 低水準関数の仕様

以下より、低水準ライブラリを作成するための、各関数の仕様を示します (アルファベット順)。内容は、形式 (関数を呼び出すためのインタフェース)、機能 (動作概要)、関数値 (値: 意味)、および解説 (動作内容や実現上の注意事項、例など) です。

5.4.1 _exit

形式

```
void _exit(int ex_num);
int ex_num; /* プログラムの終了コード */
```

機能

プログラムを終了 (exit) します。

関数値(戻り値)

なし

解説

_exit関数は、プログラムが終了することを環境に通知します。つまり、この関数内において使用している環境の終了処理を記述してください。

5.4.2 _get_core

形式

```
void *_get_core(int size);
int size; /* 割り付けるデータのサイズ */
```

機能

メモリ領域を割り付けます。

関数値(戻り値)

‘割り付けた領域の先頭アドレス’
　　正常終了

‘(void*)0’
　　エラー終了

解説

メモリ領域を割り付けるサイズが引数として渡されます。割り付けるための空き領域がなくなった場合はエラーになります。正常に割り付けができた場合は割り付けた領域の先頭アドレスを、失敗した場合は「(void *)0」を返してください。_get_core関数の実現例を以下に示します。

```
/* _get_core 関数で管理する領域のサイズをバイト数で
 * 指定してください。
 */
#define  HEAPSIZE xxxxxx

/* 割り付けのためのデータ領域の宣言です。
 * 4 バイト境界に整合させるために、int 型との共用体で
 * 宣言します。
 */
static union {
    int dummy;
    char heap [HEAPSIZE];
}heap_area;

/* 割り付ける領域への先頭
 * アドレスを代入して初期化します。
 */
static char *brk = heap_area.heap;

void *_get_core(int size)
{
    char *p;

    /* 領域が残っているかどうかチェックします。 */
    if(brk + size> heap_area.heap + HEAPSIZE)
        return((void *)0);

    p = brk;
    /* 割り付けた領域の最終アドレスを更新します。 */
    brk += size;
    /* 割り付けた領域の先頭アドレスを返します。 */
    return((void)p);
}
```

5.4.3 _rel_core

形式

```
void _rel_core(void *ptr);
void *ptr; /* 解放するメモリへのポインタ */
```

機能

_get_core関数で得られたメモリを解放します。

関数値(戻り値)

なし

解説

解放するメモリへのポインタが引数として渡されます。_get_core関数で管理しているメモリを解放する処理をします。

5.4.4 _strerror

形式

```
char *_strerror(int error_number);
int error_number; /* ユーザー定義のエラー番号 */
```

機能

ユーザー定義のエラー番号に対応するエラーメッセージを返します。

関数値(戻り値)

エラーメッセージ(文字列) へのポインタ

解説

エラー番号が引数として渡されます。この番号に対応するエラーメッセージ文字列へのポインタを返してください。

5.4.5 close

形式

```
int close(int file_no);
int file_no; /* クローズするファイル番号 */
```

機能

ファイルをクローズします。

関数値(戻り値)

'0'　　正常終了

'-1'　　エラー終了

解説

open関数で得られたファイル番号が引数として渡されます。open関数で設定したファイルの管理情報の領域を再び使用できるように解放してください。また、低水準ライブラリ内でファイルのバッファリングを行っている場合は、バッファの内容を実際のファイルに出力してください。ファイルを正常にクローズできた場合は 0、失敗した場合は-1 を返してください。

5.4.6 getuniqnum

形式

```
int getuniqnum(void);
```

機能

複数のプログラムが同時に実行するような場合、その実行単位ごとに付けられた固有の値を得ます。

関数値(戻り値)

プログラムの動作単位に固有の値を返します。プログラム動作中は、この値は変化してはなりません。

解説

プログラムが同時に複数動作するような場合、環境がその動作単位を特定するために用いている番号を返します。この値は、例えば同時に動作している２つの動作単位が同一の実行コードである場合でも、getuniqnum関数が返す値は互いに異なる必要があります。また、同一の動作単位である以上は、getuniqnum関数は必ず同じ値を返してください。なお、プログラムが同時に複数動作しないような環境である場合は、戻り値は常に同じ値でなければなりません。

5.4.7 lseek

形式

```
long lseek(int file_no, long offset, int base);
int file_no; /* 対象となるファイル番号 */
long offset;   /* 読出／書込位置を示すオフセット( バイト単位) */
int base; /* オフセットの起点 */
```

機能

ファイルの読み出し／書き込みの位置を設定します。

関数値(戻り値)

'ファイルの先頭からのオフセット(バイト単位)'
　　正常終了

'-1'　　エラー終了

解説

ファイル番号 *file_no* のファイルにおける、読み出し／書き込みを行う位置を、バイト単位で設定します。新しい位置は、第 3 引数 (*base*) によって、以下の方法で計算され設定されます。

'(a) *base* が 0 の時'
　　ファイルの先頭から *offset* バイトの位置に設定します。

'(b) *base* が 1 の時'
　　現在の位置に *offset* バイトを加えた位置に設定します。

'(c) *base* が 2 の時'
　　ファイルのサイズに *offset* バイトを加えた位置に設定します。

ファイルがコンソールやプリンタ等の対話的なデバイスの場合、新しいオフセットの値が負になった場合、また (a)、(b) でファイルのサイズを超えた場合はエラーにします。正しくファイル位置を設定できた場合は新しい読み出し／書き込み位置のファイルの先頭からのオフセットを、そうでない場合は-1 を返してください。

5.4.8 open

形式

```
int open(const char *name, int mode, int flag);
const char *name; /* ファイルのパス名を指す文字列 */
int mode; /* ファイルをオープンするときの処理の指定 */
int flag; /* ファイルをオープンするときの処理の指定( 常に 0666) */
```

機能

ファイルをオープンします。

関数値(戻り値)

'オープンしたファイル番号'
　　正常終了

'-1'　　エラー終了

解説

引数として渡されたファイルのパス名に対応するファイルを操作するための準備をします。open関数では、後で読み出し／書き込みを行うために、ファイルの種類(コンソール、プリンタ、ディスクファイル等) を決定しなければなりません。ファイルの種類は、以後 open関数で返したファイル番号を用いて読み出し／書き込みを行うたびに参照されます。第 2 パラメータ *mode* はファイルをオープンする時の処理の指定です。このデータの各ビットの意味について以下に示します。

```
bit   0  23  24  25  26  27  28  29  30  31
     +--+------+---+---+---+---+---+---+---+---+
mode |  |      | f | e | d |   |   |   | c |b,a|
     +--+------+---+---+---+---+---+---+---+---+
```

'ビット'　解説(対応するマスク値を持つシンボル)

'a(第 31 ビット)'
　　このビットが 0 の場合、ファイルを読み出し専用にオープンすることを示します。このビットが 1 の場合、ファイルを書き込み専用にオープンすることを示します。

'b(第 30 ビット)'
　　このビットが 1 で、第 31 ビットが 0 の場合、ファイルを読み出し／書き込み両用にオープンすることを示します。なお、第 30 ビットと第 31 ビットが共に 1 となることはありません。

'c(第 27 ビット)'
　　このビットが 1 の場合、次に読み出し／書き込みを行うファイル内の位置をファイルの最後に設定することを示します。このビットが 0 の場合、ファイル内の位置はファイルの先頭になります。

‘d(第 26 ビット）’
　　このビットが 1 の場合、パス名で示すファイルが存在しない場合、ファイルを新規に作成することを示します。

‘e(第 25 ビット）’
　　このビットが 1 の場合、パス名で示すファイルが存在するならば、そのファイルの内容を捨て、ファイルのサイズを 0 にすることを示します。

‘f(第 23 ビット）’
　　このビットが１の場合、ファイルを バイナリモードでオープンすることを示します。このビットが０の場合、ファイルをテキストモードでオープンすることを示します。

mode で示したファイルの処理の指定と、実際のファイルの性質が矛盾する場合はエラーにしてください。正常にファイルがオープンできた場合は、以後の `read`、`write`、`lseek`、`close` ルーチンで使用されるファイル番号(0 以上の整数) を返してください。ファイル番号と実際のファイルの対応は、低水準ライブラリで管理する必要があります。オープンに失敗した場合は-1 を返してください。

5.4.9 read

形式

```
int read(int file_no, char *buffer, unsigned int count);
int file_no;    /* 読み出しの対象となるファイル番号 */
char *buffer;    /* 読み出したデータを格納する領域へのポインタ*/
unsigned int count; /* 読み出しバイト数 */
```

機能

ファイルからのデータの読み出しを行います。

関数値(戻り値）

‘実際に読み出したバイト数’
　　正常終了

‘-1’　　エラー終了

解説

ファイル番号 *file_no* で示すファイルから *count* で示すバイト数以内のデータの読み出しを試み、実際に読み出すことのできたデータを *buffer* の示す領域に格納します。ファイルの読み出し／書き込みの位置は、読み出したバイト数だけ先に進みます。正常に読み出しができた場合は実際に読み出したバイト数を戻り地として返し、読み出しに失敗した場合は-1 を返してください。また、読み出しデータがなくファイルの終端(EOF）に到達したときは-1 でなく 0 を返してください。

file_no がターミナル (キーボード) のようなインタラクティブなデバイスの場合、読み出すことができるバイト数は、*count* で指定したバイト数より小さくなることがあります。

注意

count で示したバイト数のデータを読み出さないと終了しないような read 関数のインプリメントは正しくありません。*file_no* がキーボードのようなインタラクティブなデバイスである

場合、入力として利用可能なデータを 1 バイトでも読み出すこと (*buffer* にセット) ができたら、直ちにそのサイズを返して関数を終了してください。

5.4.10 write

形式

```
int write(int file_no, const char *buffer, unsigned int count);
int file_no;    /* 書き込みの対象となるファイル番号 */
const char *buffer; /* 書き込みデータが格納されている領域へのポインタ */
unsigned int count; /* 書き込みバイト数 */
```

機能

ファイルへのデータの書き込みを行います。

関数値(戻り値)

'実際に書き込まれたバイト数'
　　正常終了

'-1'　　エラー終了

解説

buffer の指す領域から、*count* バイトのデータを、ファイル番号 *file_no* で示すファイルへ書き込みます。ファイルの読み出し／書き込みの位置は、書き込んだバイト数だけ先に進みます。正常に書き込みができた場合は実際に書き込んだバイト数を、書き込みに失敗した場合は-1 を返してください。

6 リエントラント性

6.1 標準ライブラリ関数のリエントラント性

標準ライブラリ関数のリエントラント (再入可能) 性は、以下の表の通りです。

マルチタスク環境や割り込みハンドラにおいてリエントラント性のない標準ライブラリ関数を利用するためには、セマフォによる排他制御、タスク切り替えの禁止、割り込みの禁止などの制御をする必要があります。

関数名	リエントラント性(○: あり、×: なし)
_action_atexit	×
_exit_mem	×
_exit_stdio	×
_get_exit_code	×
_init_atexit	×
_init_base_year	×
_init_exit_environ	×
_init_mem	×
_init_stdio	×
abort	×
abs	○
acos	×
asctime	×
asin	×
assert	×
atan	×
atan2	×
atexit	×
atof	×
atoi	×
atol	×
bsearch	○(ただし、比較関数に依存)
calloc	×
ceil	×
clearerr	×
clock	ユーザー記述に依存
cos	×
cosh	×
ctime	×
difftime	○
div	○
exit	×
exp	×
fabs	×
fclose	×

feof	×
ferror	×
fflush	×
fgetc	×
fgetpos	×
fgets	×
floor	×
fmod	×
fopen	×
fprintf	×
fputc	×
fputs	×
fread	×
free	×
freopen	×
frexp	×
fscanf	×
fseek	×
fsetpos	×
ftell	×
fwrite	×
getc	×
getchar	×
getenv	ユーザー記述に依存
gets	×
gmtime	×
isalnum	○
isalpha	○
iscntrl	○
isdigit	○
isgraph	○
islower	○
isprint	○
ispunct	○
isspace	○
isupper	○
isxdigit	○
labs	○
ldexp	×
ldiv	○
localeconv	×
localtime	×
log	×
log10	×
longjmp	○

malloc	×
mblen	○
mbstowcs	○
mbtowc	○
memchr	○
memcmp	○
memcpy	○
memmove	○
memset	○
mktime	×
modf	×
perror	×
pow	×
printf	×
putc	×
putchar	×
puts	×
qsort	○（ ただし、比較関数に依存）
raise	ユーザー記述に依存
rand	×
realloc	×
remove	ユーザー記述に依存
rename	ユーザー記述に依存
rewind	×
scanf	×
setbuf	×
setjmp	○
setlocale	×
setvbuf	×
signal	ユーザー記述に依存
sin	×
sinh	×
sprintf	×
sqrt	×
srand	×
sscanf	×
strcat	○
strchr	○
strcmp	○
strcoll	○
strcpy	○
strcspn	○
strerror	○（ ただし、_strerror に依存）
strftime	×
strlen	○

strncat	○
strncmp	○
strncpy	○
strpbrk	○
strrchr	○
strspn	○
strstr	○
strtod	×
strtok	×
strtol	×
strtoul	×
strxfrm	○
system	ユーザー記述に依存
tan	×
tanh	×
time	ユーザー記述に依存
tmpfile	×
tmpnam	×
tolower	○
toupper	○
ungetc	×
va_arg	○
va_end	○
va_start	○
vfprintf	×
vprintf	×
vsprintf	×
wcstombs	○
wctomb	○

7 C 標準ライブラリの振舞い

本章では、ANSI 規格が示すところの「 未定義の振舞い(undefined behavior) 」「 インプリメンテーション依存の振舞い(implementation-defined behavior) 」および「 ロケール特有の振舞い(locale-specific behavior) 」に対する C 標準ライブラリの振舞い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、「 未定義の振舞い」、「 インプリメンテーション依存の振舞い」および「 ロケール特有の振舞い」を次のように定義しています。

未定義の振舞い
: 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振舞い。

インプリメンテーション依存の振舞い
: 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)。

ロケール特有の振舞い
: 国籍、文化、言語といった地域の習慣に依存した動作。

7.1 ANSI 規格未定義の振舞い

ANSI 規格で「 未定義の振舞い」扱いとなっている動作は、C コンパイラでの動作が保証されません。ほとんどの場合、無視する、エラーなどの警告を出す、実行時にエラーが発生する、のいずれかとなる可能性があります。したがって、「 未定義の振舞い」に該当しないようなコーディングを行うことが推奨されます。

以下より、「 未定義の振舞い」扱いの動作に対する、C 標準ライブラリで予測される振舞い(保証されるものではありません) を示します。「 ANSI 規格」に続く番号と見出しは、対応する ANSI 規格「 ANSI/ISO 9899-1990」の節番号と節タイトルです。

7.1.1 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。

7.1.2 ANSI 規格 7.1.2　標準ヘッダ

外部定義内でのインクルード
: 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、およびキーワードと同じ名前のマクロ定義は、初回参照前に、対応する標準ヘッダファイルをインクルードする必要があります。参照後にインクルードした場合、正しく動作しません。

予約外部名の再定義
: プログラム予約外部名(ヘッダ内の外部名など) を定義した場合の処理はリンカに依存します。

7.1.3 ANSI 規格 7.1.4 エラー<errno.h>

errno は、マクロおよび外部変数により実現されています。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。

7.1.4 ANSI 規格 7.1.6 共通定義<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エラーとなります。

7.1.5 ANSI 規格 7.1.7 ライブラリ関数の使用方法

ライブラリ関数の実引数が無効な値の場合、プログラムの動作は保証されません。

可変個引数を受け付けるライブラリ関数が、ヘッダのインクルードなどによって宣言されていない場合、該当関数は正しく動作しないことがあります。

7.1.6 ANSI 規格 7.2 診断関数ヘッダ<assert.h>

assert はマクロにより実現されています。関数をアクセスするためにマクロ呼び出しを無効にして assert を呼び出すと、コンパイル時にはワーニングとなり、リンク時には外部シンボルが存在しないためエラーとなります。

7.1.7 ANSI 規格 7.3 文字操作関数ヘッダ<ctype.h>

文字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。

7.1.8 ANSI 規格 7.6 大域ジャンプ関数ヘッダ<setjmp.h>

setjmp はマクロ定義を無効にした場合でもエラーとはなりません。

7.1.9 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。これら以外の用途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（ 式の評価の途中結果など）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 へのキャストなど）

7.1.10 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。

longjmp 関数がネストされたシグナルのルーチンから起動した場合、setjmp 関数に戻りますが、それ以降の動作はユーザーが作成する signal 関数（ 低水準関数）の仕様に依存します。

7.1.11 ANSI 規格 7.7.1.1 signal 関数

C 標準ライブラリには signal 関数は実装されていません。シグナルに関する処理はユーザーが作成する signal 関数(低水準関数) の仕様に依存します。したがって、次のような場合もユーザーが作成する signal 関数(低水準関数) の仕様に依存します。

- シグナルが、abort または raise 関数呼び出しの結果として発生する場合。
- シグナルハンドラが、signal 関数以外の標準ライブラリ関数を呼び出す場合。
- シグナルハンドラが、volatile sig_atomic_t 型以外の任意の静的オブジェクトを参照する場合。
- シグナル発生後 (abort、raise 関数呼び出しの結果を除く)、errno の値が参照され、対応するシグナルハンドラが値 SIG_ERR を返す signal 関数を呼ぶ場合。

7.1.12 ANSI 規格 7.8.1 可変個の実引数アクセス関数ヘッダ <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の 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／しないにかかわらず、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。

ただし、ap のアドレスを引数にして渡したり、集合体(ap が集合体であるとき) を引数として渡した場合、関数 B から戻ってきた後の関数 A の ap は関数 B の終了時点からの続きになります。

va_start、va_arg、va_end はマクロにより実現されています。関数をアクセスするためにマクロ呼び出しを無効にしてこれらを呼び出すと、コンパイル時にはワーニングとなり、リンク時には外部シンボルが存在しないためエラーとなります。

7.1.13 ANSI 規格 7.8.1.1 va_start マクロ

va_start マクロの第 2 引数 parmN の型宣言が、レジスタクラス変数、関数型、配列型であるとき、または、規定実引数拡張後の型(引数に対する暗黙の型変換後の型) と合わないとき、動作は保証されません。

7.1.14 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、動作は保証されません。

va_arg を呼び出したとき、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。

7.1.15 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 を呼び出してもエラーとはならず正常に動作します。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 してもエラーにはなりませんが、プログラムの動作は保証されません。

7.1.16 ANSI 規格 7.9.5.2 fflush 関数

入力ストリームに対する fflush は無視されます(エラーも返りません)。

7.1.17 ANSI 規格 7.9.5.3 fopen 関数

同じストリームに対する入力要求と出力要求の間に、fflush 関数またはファイル位置指定関数[1] がない場合、入出力動作は保証されません。

7.1.18 ANSI 規格 7.9.6 書式指定入出力関数

printf 系／ scanf 系

関数仕様の型と引数リストの対応する数が一致しない場合、および、変換指示子の数より引数の数が少ない場合、エラーとはなりませんが動作は保証されません。変換指示子の数より引数が多い場合、余分な引数は無視されます。

printf 系、scanf 系関数において無効な変換仕様に対する入出力結果は不定です。ほとんどの場合、エラーメッセージは出力されません。期待どおりに入出力が得られない場合、変換仕様のコーディングが正しい書式かどうか確認してください。

printf 系／ scanf 系

%%変換 printf 系または fscanf 系関数の変換仕様%%で、%と%の間に数字以外の文字含む場合、多くの場合、%と%の間の文字が入出力対象となります。例えば、"%abcdefg%" は "abcdefg" に変換されます。

7.1.19 ANSI 規格 7.9.6.1 printf 系

修飾子　printf 系の変換仕様において、修飾子(サイズ指定文字 h、l) が該当変換指定子(o,x,X,e,E,f,g,G) 以外の変換指定子の前のｈまたはｌの前に指定された場合、その修飾子は無視されます。

フラグ　printff 系の変換仕様において、フラグ#が該当変換指示子(o,x,X,e,E,f,g,G) 以外の前に指定された場合、そのフラグは無視されます。

printff 系の変換仕様において、フラグ 0 が該当変換指示子(d,i,o,u,x,X,e,E,f,g, G) 以外の前に指定された場合、そのフラグは無視されます。

変換結果　集合体、共用体、または集合体か共用体へのポインタが 、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 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、動作は保証されません。

[1] fgetpos, fseek, fsetpos, ftell, rewind

7.1.20 ANSI 規格 7.9.6.2 scanf 系

修飾子
: scanf 系の変換仕様において、修飾子(サイズ指定文字 h、l、L）が以下のように該当変換指定子以外の前に指定された場合、その修飾子は無視されます。
  - d,i,n,o,u,x 以外の変換指定子の前の h または l
  - e,f, g 以外の変換指定子の前の L

printf 系の%p との互換
: 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互換性があります。

変換結果の格納領域
: scanf 系関数による変換結果値を代入する領域が、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。

7.1.21 ANSI 規格 7.10.1 文字列変換関数(文字列から数値への変換）

文字列を数値に変換する関数(atof 、atoi 、atol）の変換結果が定義域エラーで表現できない値の場合、定義域最大値(HUGE_VAL、INT_MAX など）を返します。

7.1.22 ANSI 規格 7.10.3 メモリ管理関数(free 関数、realloc 関数）

free 関数か realloc 関数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 関数の第 1 引数(解放対象領域へのポインタ）として次のような値を渡した場合、動作は保証されません。

- calloc、malloc、または realloc 関数の戻り値(割り当て完了領域へのポインタ）でない値。
- 以前に free 関数または realloc 関数によって解放された領域へのポインタ。

7.1.23 ANSI 規格 7.10.4.3 exit 関数

プログラムが 2 回以上 exit 関数の呼び出しを実行する場合(atexit 関数で exit 関数登録した場合など）の動作は保証されません。

7.1.24 ANSI 規格 7.10.6 整数型の算術演算関数

整数型の算術演算関数(abs、div、labs、ldiv）の結果が表現できない場合、その値は保証されません。

7.1.25 ANSI 規格 7.10.7 マルチバイト文字関数(シフト状態）

マルチバイト文字のシフト状態はありません(MB_CUR_MAX=1)。したがって、LC_CTYPE 部分のシフト状態値はマルチバイト文字操作関数の動作には影響しません。

7.1.26 ANSI 規格 7.11.2,7.11.3　複製／連結関数

以下の場合、動作は保証されません。

- memcpy、memmove、strcpy、strncpy 関数において、複写先のサイズが複写元のサイズより小さい場合。
- strcat、strncat 関数において、連結される文字列の格納領域が、連結結果を格納するために十分でない場合。

7.1.27 ANSI 規格 7.12.3.5　strftime 関数

strftime 関数の時間変換書式 format 中に変換指定子でない文字がある場合、その文字をそのまま出力します。

同じ配列を指していないポインタ同士で減算する場合、全オペランド(ポインタ）の型が適合する型(compatible type）ならばエラーとはなりません。ただし、キャストによって型を適合させた場合、演算結果は保証されません。

実引数と仮引数の数の不一致
: 関数プロトタイプが見えないところでの関数呼び出しにおいて、実引数と仮引数の数が一致しない場合、不足分の仮引数の値は保証されません。

ワイド文字列リテラルの連結
: ワイド文字列リテラル指定時、連結文字列は無効となります。

左辺値によるアクセス
: 適合しない型をもつ左辺値をオブジェクトに代入した場合、動作は保証されません。

7.2 インプリメンテーション依存の振舞い

ANSI 規格で「 インプリメンテーション依存の振舞い」扱いとなっている動作について、C 標準ライブラリにおける振舞いを以下に示します。

「 ANSI 規格」に続く番号と見出しは、対応する ANSI 規格「 ANSI/ISO 9899-1990」の節番号と節タイトルです。

7.2.1 ANSI 規格 7.1.6　共通定義ヘッダ <stddef.h>

＜ NULL で展開される null ポインタ＞
: マクロ定数 NULL(null ポインタ）の定義は ((void *) 0) です。

7.2.2 ANSI 規格 7.2　診断関数用ヘッダ <assert.h>

＜ assert 関数による診断メッセージ＞
: assert 関数の終了によって出力される診断メッセージは「 Assertion failed: 式, file: ファイル名, line:行番号」です。

7.2.3 ANSI 規格 7.3.1 文字判定関数

＜文字判定関数で検査される文字セット＞

isalnum、isalpha、iscntrl、islower、isupper、isprint が真を返す文字(ASCII 文字）は次のとおりです。

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

7.2.4 ANSI 規格 7.5.1 エラー処理

＜定義域エラー(domain error）発生時、数値計算関数が返す値＞

数値計算関数において定義域エラーが起きた場合、errno に EDOM が設定されます。

＜アンダーフローエラー発生時、数値計算関数が errno を ERANGE マクロの値にセットするかどうか。>

数値計算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 が設定されます。

7.2.5 ANSI 規格 7.5.6.4 fmod 関数

＜ fmod 関数の第 2 引数が 0 のとき＞

fmod 関数の第 2 引数が 0 の場合、ドメインエラーとなり、errno に EDOM が設定されます(計算結果は保証されません)。

7.2.6 ANSI 規格 7.7.1.1 signal 関数

＜ signal 関数のシグナル設定＞

ユーザーが作成する signal 関数(低水準関数）の仕様に依存します。

＜ signal 関数で処理できるシグナルは？＞

ユーザーが作成する signal 関数(低水準関数）の仕様に依存します。

＜ signal 関数で処理できる各シグナルの、デフォルトハンドラおよびプログラム開始ハンドラ＞

ユーザーが作成する signal 関数(低水準関数）の仕様に依存します。

＜ signal(sig, SIG_DEL) 相当のものがシグナルハンドラ呼び出しより先に実行されない場合、シグナルのブロックは実行されるか。＞

ユーザーが作成する signal 関数(低水準関数）の仕様に依存します。

＜signal 関数に指定したハンドラによってシグナル SIGILL を受け付けた場合、デフォルトハンドラがリセットされるかどうか。＞

ユーザーが作成する signal 関数（ 低水準関数）の仕様に依存します。

7.2.7 ANSI 規格 7.9.2 ストリーム

＜テキストストリーム（ テキストファイル）の最終行には改行文字が必要か。＞

ユーザーが作成する read 関数および write 関数（ 低水準関数）の仕様に依存します。なお、これらの関数を呼び出す C 標準ライブラリの関数は、最終行に改行コードがなくても動作するように設計されています。

＜改行文字の直前にテキストストリームへ書き出された空白文字は読み込み時に出力されるか。＞

ユーザーが作成する read 関数および write 関数（ 低水準関数）の仕様に依存します。

＜バイナリストリームに追加される null 文字の数＞

バイナリストリームの末尾に null 文字は付加されません。

7.2.8 ANSI 規格 7.9.3 ファイル

＜追加モードストリームのファイル位置指示子の位置＞

ファイル位置指示子は、はじめにファイルの終わりに位置します。

＜テキストストリームへの書き込みがそのポイントを超えて関連ファイルを切り詰めるかどうか。＞

テキストストリームの切り詰めは起こりません。

＜ファイルバッファリングの特性＞

バッファリング方式には、フルバッファリング、行バッファリング、バッファリングなしの 3 種類があり、setbuf 関数および setvbuf 関数によって設定／変更できます。

＜長さ 0 のファイルが実際に存在するかどうか。＞

ユーザーが作成する低水準関数の仕様に依存します。

＜有効なファイル名を作るための規則＞

ユーザーが作成する低水準関数の仕様に依存します。

＜同一のファイルを何回もオープンできるか。＞

ユーザーが作成する open 関数（ 低水準関数）の仕様に依存します。

7.2.9 ANSI 規格 7.9.4.1 remove 関数

＜オープンファイルにおける remove 関数の効果＞

ユーザーが作成する remove（ 低水準関数）の仕様に依存します。

7.2.10 ANSI 規格 7.9.4.2 rename 関数

＜ rename 関数を呼び出す前に新しい名前を持つファイルがあった場合、そのファイルはどうなるか。＞

ユーザーが作成する rename 関数（ 低水準関数）の仕様に依存します。

7.2.11 ANSI 規格 7.9.6.1 fprintf 関数

＜ fprintf 関数における%p 変換の出力＞

printf 系の変換指定%p の出力は、%08X の出力と等しくなります。

7.2.12 ANSI 規格 7.9.6.2 fscanf 関数

＜ fscanf 関数における%p 変換の入力＞

scanf 系の変換指定%p の入力は、%X の入力と等しくなります。

＜ scanf 系におけるハイフンの解釈＞

scanf 系の%[変換では、—（ ハイフン）で文字コードセット上の範囲を指定し、複数の文字を指定できます。以下に、文字セットが ASCII の場合の指定例を示します。

指定	指定される文字セット
%[a-f]	'a'-'f'（ abcdef の 6 文字）
%[0-9][a-f]	'0'-'9' と'a'-'f'（ 0123456789abcdef の 16 文字）
%[–a]	'-' から'a' までの文字
%[—]	'-' のみ
%[z-a]	なし（ 対応する文字がありません）
%[-af]	'-'、'a'、および'f'（ ハイフンが 1 文字目の場合、範囲指定と見なされません）
%[af-]	'a'、'f'、および'-'（ ハイフンが最後の文字の場合、範囲指定と見なされません）
%[a-f-z]	'a'-'f' 、'-'、および'z'

7.2.13 ANSI 規格 7.9.9.1 fgetpos 関数、7.9.9.4 ftell 関数

＜ fgetpos 関数、ftell 関数が失敗した場合＞

fgetpos 関数および ftell 関数が失敗した場合（ 環境がエラーを返した場合）、errno には EFSEEK が設定されます。

7.2.14 ANSI 規格 7.9.10.4 perror 関数

＜ perror 関数によって生成されるメッセージ＞

ユーザーが作成する_strerror 関数（ 低水準関数）の仕様に依存します。

7.2.15 ANSI 規格 7.10.3 メモリ操作関数

＜ 0 バイトサイズのメモリが要求された場合の calloc、malloc、realloc 関数の動作＞

calloc、malloc、realloc 関数に 0 バイトのサイズのメモリを要求したときは、NULL ポインタを返します。

7.2.16 ANSI 規格 7.10.4.1 abort 関数

＜オープンファイルと一時ファイルに関する abort 関数の動作＞

ユーザーが作成する abort 関数（ 低水準関数）の仕様に依存します。

7.2.17 ANSI 規格 7.10.4.3 exit 関数

＜ exit 関数が返す終了ステータス（ 引数の値が 0、EXIT_SUCCESS、または EXIT_FAILURE のいずれでもない場合）＞

ユーザーが作成する_exit 関数（ 低水準関数）の仕様に依存します。

7.2.18 ANSI 規格 7.10.4.4 getenv 関数

＜ getenv 関数における環境名のセットおよび環境リストの変更方法＞

ユーザーの作成する getenv 関数（ 低水準関数）の仕様に依存します。

7.2.19 ANSI 規格 7.10.4.5 system 関数

＜ system 関数による文字列の内容および実行モード＞

ユーザーの作成する system 関数（ 低水準関数）の仕様に依存します。

7.2.20 ANSI 規格 7.11.6.2 strerror 関数

＜ strerror 関数に返されるエラーメッセージ＞

ユーザーが作成する_strerror 関数（ 低水準関数）の仕様に依存します。

7.2.21 ANSI 規格 7.12.1 時刻データの内容

＜現地時間と夏時間＞

現地時間（ 各ロケールにおける暦時間）の設定には環境変数 TZ を使用します。現地時間として、JST（ デフォルト）、EST5EDT、CST6CDT、MST7MDT、PST8PDT、UTC をサポートしています。夏時間として、EST5EDT、CST6CDT、MST7MDT をサポートしています。

7.2.22 ANSI 規格 7.12.2.1　clock 関数

＜ clock 関数における経過時間＞
　　ユーザーが作成する clock 関数（ 低水準関数）の仕様に依存します。

7.3 ロケール特有の振舞い

ANSI 規格で「 ロケールに特有な振舞い」扱いとなっている動作について、C 標準ライブラリにおける振舞いを以下に示します。

「 ANSI 規格」に続く番号と見出しは、対応する ANSI 規格「 ANSI/ISO 9899-1990」の節番号と節タイトルです。

7.3.1 ANSI 規格 5.2.1　文字セット

＜拡張実行用文字セットの内容＞
　　文字セットに拡張はありません。

7.3.2 ANSI 規格 5.2.2　文字表示の意味

＜印字方向＞
　　印字方向は左から右です。

7.3.3 ANSI 規格 7.1.1　ライブラリ用語定義

＜小数点を表す文字＞
　　小数点を表す文字は、"C"環境（ "C" locale）、""環境（ "" locale）とも 0x2E（ '.'）です。

7.3.4 ANSI 規格 7.3　文字操作関数用ヘッダ<ctype.h>

＜文字判定関数における判定文字セット＞
　　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、「 インプリメンテーション依存の振舞い」の「 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。

　　""環境（ "" locale）では、ctype.h に定義および宣言されたマクロや関数は、"C" 環境（ "C" locale）と同じ動作をします。

7.3.5 ANSI 規格 7.11.4.4　strncmp 関数

＜文字セットの照合順序 ＞
　　strncmp 関数における文字セットの照合順序は ASCII の照合順序と同じです。使用される文字セットは、""環境（ "" locale）、"C"環境（ "C" locale）ともに同じです。

7.3.6 ANSI 規格 7.12.3.5 strftime 関数

＜時刻と日付の形式＞

strftime 関数における日付および時刻は、"C"環境("C" locale）、""環境("" locale）ともに次のように表現します。

日付	mm1/dd/yy（ mm1 は月、dd は日、yy は西暦の下 2 桁）
時刻	hh:mm2:ss（ hh は時間、mm2 は分、ss は秒）

技術サポート連絡書

年　月　日（合計　　枚）

三菱電機セミコンダクタシステム株式会社
マイコンソフトツール部
開発ツールサポート窓口行

［電子メール］ support@tool.mesc.co.jp
［大阪地区］ FAX： 06-6398-6191
［東京地区］ FAX： 03-5783-7339
［中部地区］ FAX： 052-221-7318
［九州地区］ FAX： 092-452-1427

※インストーラが生成する以下のテキストファイルもサポート連絡書としてご利用できます。
Windows 98/95/Windows NT 4.0版の場合：\SUPPORT\製品名\SUPPORT.TXT
EWS版の場合：/support/製品名/toolinfo.txt

ご連絡先	製品情報
会社名：	ソフトウェア：
部署名：	バージョン番号：V.
担当者名：	ライセンスID：
電話番号：	– – – –
FAX番号：	ホストマシン：
電子メール：	OS： V.
通信欄：	

MEMO

TW32R V.3.10標準ライブラリマニュアル

第1版：2000年6月1日発行
資料番号：MSD-TW32R-US-000601

三菱電機株式会社
三菱電機セミコンダクタシステム株式会社

TW32R V.3.10 標準ライブラリマニュアル
M32R ファミリ用　クロスツールキット (GNU 版)

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668