

SHARP

**SHARP
COMPUTER
SOFTWARE**

 **68000用**

C **COMPILER**  **ver2.0**  **PRO-68K**

Cライブラリマニュアル VOL.1

 68000用

COMPILER  ver2.0
PRO-68K

Cライブラリマニュアル VOL.1

SHARP

はじめに

本書は、「C compiler PRO-68K ver2.0」(以下、XC コンパイラと表記します)でプログラム開発を行うときに必要となる C ライブラリの概要と使用方法、小文字関数リファレンスに関して記述しています。

大文字関数のリファレンスに関しては、「C ライブラリマニュアル VOL.2」をご覧ください。

なお、別冊の「C ユーザーズマニュアル」に XC コンパイラの商品構成、動作環境の作成方法、各マニュアルの内容が説明されています。

初めて本プログラムをご使用になる方は、必ず「C ユーザーズマニュアル」を先にご覧ください。

本書は、C 言語について一定以上の知識を有し、プログラミング経験のあるユーザーが、C ライブラリの機能と使用方法を理解して X68000 の豊富な機能をフルに活用できるようになること、さらに進んでユーザーが自分自身でもライブラリを作成できるようになることを目的として書かれています。

なお、C 言語の書式・文法については「C リファレンスマニュアル」、開発環境の設定や開発手順などについては「C ユーザーズマニュアル」を参照してください。

Cライブラリの概要

Human68k の C ランタイムライブラリは、C 言語でプログラム開発を行う際に必要となる関数およびマクロで構成されています。

C 言語では、データの入出力などハードウェアに依存する処理は、言語仕様からはずし、これを関数として提供しています。

また、文字列操作、およびサーチやソートといった、使用頻度の高い汎用的な処理についても関数として提供し、C プログラマの労力を軽減しています。

Human68k の C ランタイムライブラリでは、上記のような C 標準ライブラリの他に、日本語処理をサポートする関数や、IOCS コールライブラリ、DOS コールライブラリ、BASIC ライブラリといった、Human68k 固有の関数が豊富に用意され、その処理能力はきわめて強力なものとなっています。

たとえば、IOCS コールライブラリによって、キーボードやプリンタといった標準的な入出力機器だけでなく、ジョイスティック、RS-232C、ADPCM、FM 音源、マウス、グラフィック、スプライト機能といった、多様な入出力操作が可能となります。

また、DOS コールライブラリによって、日本語処理、日付・時刻の制御、環境変数の操作などの、Human68k オペレーティングシステムの機能を有効に利用することができます。

BASIC ライブラリは、BASTOC において使用し、BASIC 言語から C 言語への変換を容易にしています。

また、これらの関数をサポートするために、専用のインクルードファイルが用意されています。

C ランタイムライブラリの項目別の分類、およびその項目に属する関数の名前と機能については、「第 3 章 項目別ランタイムルーチン」を参照してください。

本書の構成

本書は以下の内容から構成されています。

第1章 ライブラリルーチンの使いかた

ライブラリルーチンの基本的な使い方、および関数とマクロの違い、ライブラリとともに使用するインクルードファイル、エラー処理など、各ライブラリに共通する項目について述べます。

第2章 インクルードファイル

ライブラリ使用時に必要となるインクルードファイルの概要、およびその使用方法について述べます。

また、Cライブラリで使用する型とマクロ、グローバル変数および、標準型の構造体についても述べます。

第3章 項目別ランタイムルーチン

Cライブラリを項目別に分けて、ライブラリの種類とその機能について説明します。

第4章 リファレンス（小文字）

各ライブラリのリファレンスであり、実際にライブラリを使用するときには使用手引として利用してください。

付録

ユーザーライブラリの作成方法を、具体的な例にそって説明します。

CONTENTS

第1章 ライブラリルーチンの使いかた

1.1 関数とマクロ	3
1.2 インクルードファイル	4
1.3 エラーハンドリング	5
1.4 ライブラリファイル	6

第2章 インクルードファイル

2.1 インクルードファイルについて	9
2.2 型とマクロの定義	12
2.3 グローバル変数	15
2.4 標準型構造体名	16
2.5 インクルードファイルの種類	17
2.5.1 assert. h	19
2.5.2 audio. h	19
2.5.3 basic. h	19
2.5.4 basic0. h	19
2.5.5 class. h	19
2.5.6 conio. h	20
2.5.7 ctype. h	20
2.5.8 direct. h	20
2.5.9 doslib. h	20
2.5.10 error. h	20
2.5.11 fcntl. h	20
2.5.12 fctype. h	21
2.5.13 float. h	21
2.5.14 graph. h	21
2.5.15 image. h	21
2.5.16 io. h	21
2.5.17 iocslib. h	21
2.5.18 jfctype. h	22
2.5.19 jstring. h	22
2.5.20 limits. h	22
2.5.21 math. h	22
2.5.22 mouse. h	22
2.5.23 music. h	22

2.5.24	music2. h	23
2.5.25	process. h	23
2.5.26	setjmp. h	23
2.5.27	signal. h	23
2.5.28	sprite. h	23
2.5.29	stat. h	23
2.5.30	stdarg. h	24
2.5.31	stddef. h	24
2.5.32	stdio. h	24
2.5.33	stdlib. h	25
2.5.34	stick. h	25
2.5.35	string. h	25
2.5.36	time. h	26
2.5.37	timeb. h	26
2.5.38	utime. h	26

第3章 項目別ランタイムルーチン

3.1	バッファ操作	29
3.2	メモリ割りあて	31
3.3	文字の分類と変換	33
3.4	データ変換	35
3.5	ディレクトリ制御	37
3.6	ファイル操作	38
3.7	ストリーム入出力	40
3.8	低水準入出力	47
3.9	コンソール入出力	50
3.10	数値演算	51
3.11	プロセス制御	53
3.12	サーチとソート	56
3.13	文字列操作	57
3.14	時 間	59
3.15	日本語ライブラリ	61
3.15.1	文字分類関数	61
3.15.2	漢字処理関数	61
3.16	BASIC ライブラリ	64
3.17	IOCS コール	72

CONTENTS

3.18 DOS コール	82
3.19 その他	92

第4章 リファレンス

.....	93
-------	----

付 録 ユーザーライブラリの作り方

1. ソースファイルの作成	567
2. コンパイル	568
3. ライブラリファイル	569
4. ライブラリの外部宣言	570
5. リンク	571

索 引

50 音順	573
アルファベット順	580

関数索引

.....	584
-------	-----

機能別関数索引

.....	597
-------	-----

第1章

ライブラリルーチンの 使い方

この章では、ライブラリルーチンの使い方について、関数とマクロ、インクルードファイル、エラーハンドリング、ライブラリファイルの扱い方を説明します。

関数とマクロ

インクルードファイル

エラーハンドリング

ライブラリファイル

第1章

モーターリセット の仕方

本章では、Cランタイムライブラリを使用するにあたって注意しなければならないこと、ライブラリの使用環境、ライブラリの設定方法などについて説明します。

関連する項目

モーターリセット
の仕方
モーターリセット
の仕方

1.1 関数とマクロ

C ランタイムライブラリは、C 言語、あるいはアセンブラで作成された関数であり、一般の関数と同じようにプログラムの中から呼び出すことができます。

同時に、これらのライブラリのうちのいくつかは、マクロとしても用意されています。

マクロとは、プリプロセッサ命令 `#define` で定義する、値や式を表す識別子のことです。

関数もマクロも引数をまったくとらないか、または、マクロ定義で仮引数に置き換え可能な引数を定義することができます。

いいかえれば、マクロも関数と同じように、引数を取り、値を返すことができます。

関数とマクロの違いは、実行速度とメモリサイズという点にあります。

たとえば、マクロはコンパイル前にプリプロセッサによって展開されるので、関数呼び出しの際にみられるようなオーバーヘッドはありません。

しかし、マクロは一度しか定義されない関数とは異なり、呼び出されるたびに展開されるので、必然的にメモリサイズが大きくなります。

このため、C ランタイムライブラリでは特性に応じて選択できるように、関数定義とマクロ定義の双方をもつライブラリルーチンがいくつかあります。

なお、マクロの中には、`toupper` ルーチンのように、引数を2回以上評価する場合、副作用が発生して結果が正しく返らないものもありますので注意してください (`toupper` ルーチンの使用上の注意については、「3.3 文字の分類と変換」および `toupper` ルーチンのリファレンスを参照してください)。また、使用にあたって注意が必要となるものについては、そのつど説明します)。

1.2 インクルードファイル

Cランタイムライブラリの関数、またはマクロを呼び出すにあたっては、それらの関数やマクロを外部宣言し、(関数やマクロで使用する) 構造体や記号定数を定義しているインクルードファイルが必要となります。

必要となるインクルードファイルは、「第3章 項目別ランタイムルーチン」または、「第4章 リファレンス」に記載していますので、これに従い、`#include` プリプロセッサ命令を使ってソースファイルにとり込んでください。

1.3 エラーハンドリング

C ランタイムライブラリの関数のほとんどは戻り値を返しますが、この戻り値によって、処理が正常に終了したか、あるいは異常終了したかが判定できるようになっています。

関数を呼び出したときは必ず戻り値の判定を行うようにしてください。関数が異常終了したにもかかわらず、戻り値の判定を行わずに処理を続行した場合、プログラムが暴走することもあります。

C ランタイムライブラリの関数の中には、エラーが発生した場合にグローバル変数 `errno` にエラーの詳細コードを設定してくれるものもあります。このような関数を使用するにあたっては、インクルードファイル `error.h` をソースファイルにとり込むことにより、`errno` を参照し、エラーコードのチェックを行うことができます。

また、C ランタイムライブラリの中には、エラー処理を行う関数がありますので、これを以下に示します。

- `perror` 関数 エラーメッセージを標準エラー出力 (`stderr`) に表示します。
- `matherr` 関数 数値演算で発生したエラーの処理を行うため数値演算関数自身が呼び出します。
また、ユーザー自身で `matherr` 関数を作成し、使用することができます。
その場合には、`matherr` 関数のリファレンスに明記してある規約に従ってください。
- `ferror` 関数 ストリーム入出力におけるエラーをチェックします。
- `clearerr` 関数 ストリームのエラーインジケータおよびエンドオブファイルインジケータをクリアします。

1.4 ライブラリファイル

ライブラリファイルには、オブジェクトファイル、ライブラリファイル、およびアーカイブファイルという3つのファイル形式があります。

オブジェクトファイルは、Cコンパイラによって出力される1個のオブジェクトコードを格納しておくファイルであり、拡張子はoです。

これに対してライブラリファイル、およびアーカイブファイルとは、複数個のオブジェクトファイル、またはソースファイルを1個のファイルにまとめて登録したものであり、拡張子はlまたはaとなります(ただし、1個のアーカイブファイルに、オブジェクトファイルとソースファイルを混在させて登録することはできません)。ライブラリファイルやアーカイブファイルの詳細については、「アセンブルマニュアル」の「第1部 第8章 ライブラリアン」と「第1部 第7章 アーカイバ」をそれぞれ参照してください。なお、オブジェクトファイルは、ライブラリアン (LIB) を使用してライブラリファイル(拡張子l)へ、ソースファイルはアーカイバ (AR) を使用してアーカイブファイル(拡張子a)へ登録します。

なお、XCの標準ライブラリの登録ファイル名はCLIB.lです。

第 2 章

インクルードファイル

インクルードファイルについて

型とマクロの定義

グローバル変数

標準型

インクルードファイルの種類

第2章

ポインタとグローバル変数

ポインタとグローバル変数

本章では、インクルードファイルについて説明します。

フォーマットの定まった変数や構造体が、あらかじめ定義されているファイルをインクルードファイルといいます。

XCで特定されている、このインクルードファイルにおける、グローバル変数と呼ばれる変数、標準型と呼ばれる構造体名、ならびにインクルードファイルの種類についても解説します。

2.1 インクルードファイルについて

Cのプログラムを作成するにあたって、ユーザーは必要な処理手順、変数等をコーディングしなければなりません。たとえば、プログラム間で共通に使用される（フォーマットの定まった）構造体や、変数をユーザープログラムでいちいち定義しなければならないとしたら、プログラム開発の生産性が低下するだけでなく、コーディングミスの多発、といった事態も避けられません。

このようなわずらわしい作業から、プログラマを解放するためにXCでは、上記のような構造体や、変数をあらかじめ特定のファイル中に定義してあります。

このようなファイルのことを、インクルードファイルといいます。なお、インクルードファイルの拡張子は習慣的にhにします。ユーザーは、特定の構造体や変数が必要になった時点でこれをコーディングする代わりに、該当の構造体や変数が定義されているインクルードファイルを、ソースプログラムから呼び出します。

また、ライブラリの中には、これを呼び出すにあたって特定のインクルードファイルが必要となるものもあります。

Cランタイムライブラリ関数の宣言には引数型リストを含むものと、含まないものの2種類があります。

関数の宣言は、プリプロセッサの`#ifdef`、または`#ifndef`に含まれています。

XCはデフォルトでは、引数型リストを含む関数宣言が生成、コンパイルされ、引数型チェックが行われます。

（`__STDC__`識別子が1に定義されています）

2.1 インクルードファイルについて

引数型のチェックを行わないようにするには、次の方法があります。

1. CC コマンドラインの X スイッチを使用して、`__STDC__` 識別子を未定義にします。
2. ソースプログラム中で、`#undef` プリプロセッサ命令を使用して `__STDC__` の定義を取り消します。
この場合、`#undef` は `#include` の前になければなりません。
インクルードファイルの呼び出しは、`#include` プリプロセッサ命令により行います。

呼び出し手順を以下に示します。

- ① `#include` <ファイル名>
- ② `#include` "ファイル名"

①の場合は以下の順で検索します。

環境変数 `include` が指すディレクトリ
↓
カレントドライブのカレントディレクトリ下の `include` サブディレクトリ
↓
カレントドライブのカレントディレクトリ

②の場合は

カレントドライブのカレントディレクトリ
↓
環境変数 `include` が指すディレクトリ
↓
カレントドライブのカレントディレクトリ下の `include` サブディレクトリ

また、ファイル名は大文字・小文字を区別しません。
なお、ユーザー用のインクルードファイルをユーザー自身で作成することもできます（作成はエディタによって行います）。

基本のC言語

2.1 インクルードファイルについて

インクルードファイルの呼び出し例を以下に示します。

例： #include <stdio. h>

#include <ctype. h>

管端： #include "mylib. h"

```
main(int argc, char * argv [] )
```

```
{
```

```
.....
```

```
}
```

型	ファイル	内容
div_t	stdlib. h	intの剰余を求めるdiv関数の返り値を格納する型
ldiv_t	stdlib. h	longの剰余を求めるldiv関数の返り値を格納する型
ptrdiff_t	stdlib. h	ポインタ同士の間を算出した結果を正しく格納できる型
size_t	stdlib. h	バイト数やsizeofの返す値を正しく格納できる型
time_t	stdlib. h	時間やtime関数などの返す値を正しく格納できる型
clock_t	stdlib. h	秒数やclock関数などの返す値を正しく格納できる型
lpos_t	stdlib. h	fseek関数などに与えるファイルの位置を返す値を正しく格納できる型
INT	class. h	int型
UINT	class. h	unsigned int型
WORD	class. h	short型
UWORD	class. h	unsigned short型
BYTE	class. h	char型
UBYTE	class. h	unsigned char型
VOID	class. h	void型
DEFAULT	class. h	int型

2.2 型とマクロの定義

インクルードファイルには、頻繁に使用される値や、動作環境や機種に依存するような型を定義してあります。

ここで定義してある型やマクロを使用すると、ソースファイルを他のCコンパイラに移植した場合でも、変更は必要最小限で済むようになります。

定義されている型の種類とその内容は以下の通りです。

型	ファイル	内 容
div_t	stddef. h	intの剰余を求めるdiv関数の返り値を格納できる型
ldiv_t	stddef. h	longの剰余を求めるldiv関数の返り値を格納できる型
ptrdiff_t	stddef. h	ポインタ同士の引き算の結果が正しく格納できる型
size_t	stddef. h	バイト数やsizeofの返す値が正しく格納できる型
time_t	stddef. h	時間やtime関数などの返す値が正しく格納できる型
clock_t	stddef. h	秒数やclock関数などの返す値が正しく格納できる型
fpos_t	stddef. h	fgetpos関数などに与えるファイルの位置を表す値を正しく格納できる型
INT	class. h	int型
UINT	class. h	unsigned int型
WORD	class. h	short型
UWORD	class. h	unsigned short型
BYTE	class. h	char型
UBYTE	class. h	unsigned char型
VOID	class. h	void型
DEFAULT	class. h	int型

2.2 型とマクロの定義

型	ファイル	内 容
BOOLEAN	class. h	int型
LONG	class. h	long型
ULONG	class. h	unsigned long型
jmp_buf	setjmp. h	setjmp関数やlongjmp関数で使用するプログラム環境を格納できる型
va_list	stdarg. h	va_startマクロ関数やva_argマクロ関数で使用する、可変引数を指し示す変数の型
FILE	stdio. h	ストリーム入出力用の構造体を表す型

定義されているマクロの種類とその内容は以下の通りです。

マクロ	ファイル	内 容
CLK_TCK	stddef. h	clock関数が返す値から秒にする場合に割り算する値
CLOCKS_PAR_SEC	stddef. h	clock関数が返す値から秒にする場合に割り算する値
SEEK_SET	stddef. h	fseek関数で、先頭からシークする場合のモード
SEEK_CUR	stddef. h	fseek関数で、現在位置からシークする場合のモード
SEEK_END	stddef. h	fseek関数で、最後からシークする場合のモード
NULL	stddef. h	ヌルポインタを表す値
EOF	stddef. h	ストリーム入出力関数がエラーの時返す値
EOS	stddef. h	文字列の終わりを表す文字
EXIT_SUCCESS	stdlib. h	プログラムが正常終了する場合にexit関数に指定する値
EXIT_FAILURE	stdlib. h	プログラムが異常終了する場合にexit関数に指定する値
BUFSIZ	stdio. h	ファイル入出力バッファのバイト数

2.2 型とマクロの定義

マクロ	ファイル	内 容
_NFILE	stdio. h	ファイル入出力用の構造体の配列の要素数 標準で5つのストリームがオープンされているのでユーザーが同時にオープンできるストリームの個数はこの値から5を引いた個数になります
FOPEN_MAX	stdio. h	ファイル入出力用の構造体の配列の要素数 標準で5つのストリームがオープンされているのでユーザーが同時にオープンできるストリームの個数はこの値から5を引いた個数になります
TMP_MAX	stdio. h	tmpfile関数などで作成できるテンポラリファイルの最大数
L_tmpnam	stdio. h	tmpname関数などで作成されるファイル名の長さ
FILENAME_MAX	stdio. h	ドライブ名、パス名などを付加したファイル名の最大

2.3 グローバル変数

インクルードファイルには、グローバル変数と呼ばれる、各プログラムから共通に使用される変数が外部宣言されています。

グローバル変数の種類とその内容は以下の通りです。

グローバル変数	ファイル	内 容
time_t timezone	time. h	グリニッジ標準時と地方時の時差が秒数で格納されている 日本では-32400 (-9 時間) になっている
int daylight	time. h	夏時間を採用している地域では非0が格納されている 日本では0が格納されている
char * tzname [2]	time. h	時刻地域名と夏時間名を示す文字列へのポインタの配列 文字列の長さは3文字 日本ではtzname [0] は"JST"、tzname [1] は""が定義されている
char tzstn []	time. h	時刻地域名が格納されている
char tzdtn []	time. h	夏時間名が格納されている
unsigned char ** environ	stdlib. h	環境文字列の配列へのポインタが格納されている
int _fmode	stdlib. h	ファイルの変換モードのデフォルト値が格納されている
unsigned int _PSP	stdlib. h	自分自身のプロセス管理ポインタの先頭のアドレスが格納されている
volatile int errno	stdlib. h	エラーの種類を表す数値が格納されている
int sys_nerr	stdlib. h	sys_errlist配列の要素数が格納されている
char * sys_errlist []	stdlib. h	エラーメッセージ文字列の配列
volatile int _doserrno	stdlib. h	システムコールでエラーが発生した場合にそのエラーの番号が格納される

2.4 標準型構造体名

インクルードファイルでは標準型と呼ばれる構造体を宣言しています。
これらの構造体名とその説明を以下に示します。

標準型	内 容
complex	complex構造体はmath. hで宣言しており、複素数の実数部と虚数部を格納する
exception	exception構造体はmath. hで宣言しており、数値演算ルーチンのエラー情報を格納する
stat	stat構造体はstat. hで宣言しており、statまたはfstat関数が返すファイルの詳細情報を格納する
tm	tm構造体はtime. hで宣言しており、asctime、gmtime、localtimeの関数が時間の情報を読み書きするために使用する
utimbuf	utimbuf構造体はutime. hで宣言しており、utime関数でファイルを修正した日付を変更するために使用する
timeb	timeb構造体はtimeb. hで宣言しており、ftime関数で使用する

2.5 インクルードファイルの種類

XCで提供しているインクルードファイルには以下のものがあります。

ファイル名	内 容
assert. h	実行時チェックマクロの定義
audio. h	ADPCMの制御関連の関数の宣言 (BASIC→Cの変換時に使用)
basic. h	X-BASIC本体組み込みの関数の宣言 (BASIC→Cの変換時に使用)
basic0. h	画面処理関係他の関数の宣言 (BASIC→Cの変換時に使用)
class. h	データ型式の定義
conio. h	コンソール入出力を行う関数の宣言
ctype. h	文字チェックおよび文字変換のマクロ定義
direct. h	ディレクトリ操作の関数宣言
doslib. h	Human68kのファンクションコールの関数宣言
error. h	エラーコードの宣言
fcntl. h	低水準入出力関数用定数の宣言
fctype. h	ctype. hで定義されているマクロの関数版の宣言
float. h	(算術ライブラリの)浮動小数点演算用定数の宣言
graph. h	グラフィック制御関連の関数の宣言 (BASIC→Cの変換時に使用)
image. h	イメージ制御関連の関数の宣言 (BASIC→Cの変換時に使用)
io. h	ファイル操作と低水準入出力関数の宣言
iocslib. h	X68000のIOCSコールの関数宣言

2.5 インクルードファイルの種類

ファイル名	内 容
jfctype. h	日本語ライブラリ用関数の宣言
jstring. h	
limits. h	各データ形式のリミットチェック
math. h	算術ライブラリ用関数の宣言
mouse. h	マウスの制御関連の関数の宣言 (BASIC→Cの変換時に使用)
music. h	FM音源の制御関連の関数の宣言 (BASIC→Cの変換時に使用)
music2. h	MIDI対応FM音源の制御関連の関数の宣言 (BASIC→Cの変換時に使用)
process. h	プロセス制御関数の宣言
setjmp. h	setjmp関数用構造体などの宣言
signal. h	signal関数用定数などの宣言
sprite. h	スプライトの制御関連の関数の宣言 (BASIC→Cの変換時に使用)
stat. h	statおよびfststat関数用定数などの宣言
stdarg. h	可変引数をあつかうためのマクロ定義
stddef. h	汎用データ型、汎用マクロ定義
stdio. h	標準入出力関数などの宣言
stdlib. h	標準ライブラリ関数の宣言
stick. h	ジョイスティック入力用の関数の宣言 (BASIC→Cの変換時に使用)
string. h	文字列関数用文字列操作関数の宣言
time. h	time関数用構造体などの宣言
timeb. h	ftime関数用構造体などの宣言
utime. h	utime関数用構造体などの宣言

2.5 インクルードファイルの種類

2.5.1 assert. h

論理エラーのチェックを行うassertマクロを定義します。
assertを使うときは必ずインクルードします。

もし、NDEBUGが定義されていないときassertにはassert関数がマクロ展開されます。

NDEBUGが定義されますとassertは空となります。

すなわち、ソースファイルからassertを取り除いたのと同じになります。

2.5.2 audio. h

ADPCMの録音および再生の機能を実行する関数を宣言しています。

なお、このインクルードファイルはBASIC→Cの変換時に使用します。

2.5.3 basic. h

X-BASIC本体組み込みの関数の宣言をしています。

おもな内容はファイル処理、数値演算、文字列処理などです。

なお、このインクルードファイルはBASIC→Cの変換時に使用します。

2.5.4 basicO. h

X-BASICの画面処理関係他の関数の宣言をしています。

主な内容は画面のクリア、画面サイズの設定、キー入力などです。

なお、このインクルードファイルはBASIC→Cの変換時に使用します。

2.5.5 class. h

データ形式(char、int等)を示すシンボルを定義しています。

このシンボルを使うことにより、他のCコンパイラで処理系によって異なる標準の型の違い(int語長など)を吸収し、移植性のよいプログラムが作成できます。

2.5 インクルードファイルの種類

2.5.6 conio. h

コンソール入出力を行う関数を宣言しています。

2.5.7 ctype. h

文字チェックや文字の変換を行うマクロ、およびこれらのマクロで使用するフラグを定義しています。

2.5.8 direct. h

ディレクトリ操作を行う関数を宣言しています。

2.5.9 doslib. h

Human68kのファンクションコールの関数の宣言をしています。これらの関数は、Human68kのファンクションコールを呼び出すもので、Cのライブラリで下から2番目（IOCSの上）に位置する関数です。よって、上位の関数と組み合わせて使う場合、注意しないと上位の関数の動作がおかしくなるおそれがあります。

2.5.10 error. h

エラーナンバーの定義。
ANSI規格のerrno. hと同じ意味で使用されます。

2.5.11 fcntl. h

低水準入出力関数（open、close等）を宣言するとともに、これらの関数を呼び出すときに使用するフラグや、ファイル種別、マクロなどを定義しています。

2.5 インクルードファイルの種類

2.5.12 fctype. h

ctype. hで定義しているマクロを、関数として宣言しています。

2.5.13 float. h

浮動小数点データの範囲などを定義しています。

2.5.14 graph. h

グラフィック制御で使用する関数を宣言しています。
このインクルードファイルは、BASIC→Cの変換時に使用します。

2.5.15 image. h

イメージ処理を行うときに必要となる関数を宣言しています。
なお、このインクルードファイルは、BASIC→Cへの変換時に使用します。

2.5.16 io. h

低水準入出力操作の関数を宣言しています。

2.5.17 iocslib. h

X68000のIOCSコールの関数を宣言しています。

C言語でこれらの関数を呼び出すことにより、X68000のハードウェアを、直接制御することができます。

これらの関数は、Cライブラリでもっとも下位に位置する関数ですので上位の関数と組み合わせて使う場合、注意しないと上位の関数の動作がおかしくなるおそれがあります。

2.5 インクルードファイルの種類

2.5.18 jfctype. h

日本語対応版文字分類関数、および文字変換関数を宣言しています。

2.5.19 jstring. h

漢字を含む文字列を操作する関数を宣言しています。

2.5.20 limits. h

整数型と文字型のデータの範囲チェックを行うために、これらの型の最大値と最小値を定義しています。

2.5.21 math. h

数学ライブラリを宣言するとともに、マクロおよび、これらのライブラリで使用するシンボルを定義しています。
定義されたシンボルは変更してはいけません。

2.5.22 mouse. h

マウスを使用するときに必要となる関数を宣言しています。

なお、このインクルードファイルは、BASIC→Cの変換時に使用します。

2.5.23 music. h

FM音源の制御に際して必要となる関数を宣言しています。

なお、このインクルードファイルは、BASIC→Cの変換時に使用します。

2.5 インクルードファイルの種類

2.5.24 music2. h

MIDI対応FM音源の制御に際して必要となる関数を宣言しています。

なお、このインクルードファイルは、BASIC→Cの変換時に使用します。

2.5.25 process. h

プロセス制御関数（signalを除く）を宣言しています。

2.5.26 setjmp. h

setjmp関数を使用するときに必要な構造体の定義や関数の宣言をしています。

2.5.27 signal. h

signal関数を使用するときに必要なシンボルを定義しています。

2.5.28 sprite. h

スプライト制御において使用する関数を宣言しています。

なお、このインクルードファイルは、BASIC→Cへの変換時に使用します。

2.5.29 stat. h

stat、fstat関数で返される構造体の定義、ファイルのステータス情報のフラグ定義、およびstat、fstatの関数宣言をしています。

2.5 インクルードファイルの種類

2.5.30 stdarg. h

printf関数などのように、引数の数が不定個の関数を作成したときに、引数の取り出しを行うためのマクロ (va_start、va_arg、va_endなど) を定義しています。

2.5.31 stddef. h

頻繁に使用される値や、動作環境や機種で異なる部分を吸収するためのシンボルなどを定義しています。

ここで定義されたシンボルを使うことにより、移植性のよいプログラムが作成できます。

EOF

ファイルの終わり、もしくはエラーが見つかったときこの値が返されます。

NULL

ヌルポインタの値を表します。

2.5.32 stdio. h

ストリーム入出力のさいに使用する構造体 (FILE)、シンボル (NULL、EOFなど)、マクロ (getchar、putchar、ferrorなど)、バッファサイズなどを定義しています。

stdio. hファイル中で定義している定数は、各プログラム中で共通に使用できますが、定義された値を変更してはいけません。

以下に、よく使われる定数の名前とその内容を説明します。

BUFSIZ

ストリームI/Oで使用するバッファのサイズを表します。

この値は、システムがバッファを確保するとき、もしくはsetbuf関数で別の

2.5 インクルードファイルの種類

バッファを確保するときに参照されます。

_NFILE

1度にオープンできるファイルの数を表します。

stdin、stdout、stderr、stdaux、stdprnの5個のファイルはつねにオープンされています。

FILE構造体は、stdio.hで定義されています。

ストリームルーチンは、ストリームをアクセスするのにFILE構造体のポインタを使います。

2.5.33 stdlib.h

標準ライブラリを宣言しています。

このインクルードファイルに宣言されているライブラリを機能別に分けると以下のようになります。

- メモリ割り当て
- ソート機能
- データ変換
- その他

2.5.34 stick.h

ジョイスティック入力時に必要となる関数(stick、strig)を宣言しています。

2.5.35 string.h

文字列操作関数やメモリ操作関数を宣言するとともに、これらの関数を使用するときに必要となるシンボルを定義しています。

2.5 インクルードファイルの種類

2.5.36 time. h

time関数を宣言するとともに、これらの関数を使用するときに必要となる構造体やシンボルを宣言しています。

2.5.37 timeb. h

ftime関数を宣言するとともに、timeb構造体を宣言しています。

2.5.38 utime. h

utime関数を定義するとともに、utimbuf構造体を宣言しています。

- 関数
- 変数
- 構造体
- シンボル

2.5.39 stick. h

stick関数を宣言するとともに、stick構造体を宣言しています。

2.5.40 string. h

string関数を宣言するとともに、string構造体を宣言しています。

第3章

項目別ランタイム ルーチン

バッファ操作/メモリ割り当て/文字の分類と変換

データ変換/ディレクトリ制御

ファイル操作/ストリーム入出力

低水準入出力/コンソール入出力

数値演算/プロセス制御

サーチとソート/文字列操作

時 間/日本語ライブラリ

BASIC ライブラリ/IOCS コール

DOS コール/その他

BASIC ライブラリ、IOCS コール、DOS コールなどは、関数名とその機能を表組みで分かりやすく解説します。

3.1 バッファ操作

バッファ操作関数は、バッファと呼ばれる特定のメモリブロック、またはメモリブロック間において、文字の検索、データのコピー・比較などを行います。バッファ操作関数の宣言は、インクルードファイルstring.hで行われています。この関数には、バッファ操作関数と拡張バッファ操作関数の2種類があり、バッファ操作関数は他のCコンパイラでも使用できる標準的な関数であり、拡張バッファ操作関数はXCで拡張された関数です。

内訳はそれぞれ以下の通りです。

バッファ操作関数

関数名	機 能
memchr	指定されたバイト数のバッファ内から、指定された文字を検索する
memcmp	2つのバッファを先頭から指定文字数分比較する
memcpy	あるバッファから他のバッファへ、指定した数の文字列をコピーする
memmove	あるバッファから他のバッファへ、指定した文字数分をコピーする
memset	バッファの先頭から指定バイト数分、指定した文字で初期化する

拡張バッファ操作

関数名	機 能
memccpy	指定された文字が現れるまでバッファ内の文字列を他のバッファにコピーする
memcmpi	2つのバッファを大文字／小文字の区別なしで先頭から指定文字数分比較する
memicmp	2つのバッファを大文字／小文字の区別なしで先頭から指定文字数分比較する

3.1 バッファ操作

関数名	機 能
movedata	あるバッファから他のバッファへ、指定した文字数分をコピーする
movmem	あるバッファから他のバッファへ、指定した文字数分をコピーする
repmem	メモリブロック中の指定箇所を複数回コピーする
setmem	バッファの先頭から指定バイト数分、指定した文字で初期化する
swmem	2つのメモリブロックの内容を入れ替える

3.2 メモリ割りあて

メモリ割りあて

メモリ割りあて関数は、メモリ領域の確保、解放、再割りあてを行います。なお、これらの関数のうち `sbrk` は低水準メモリ割りあてルーチンで、割りあてられていないメモリ領域を利用可能とします。

メモリ割りあて関数には以下のものがあり、これらの関数はすべてインクルードファイル `stdlib.h` で宣言されています。

関数名	機 能
<code>allmem</code>	使用可能なすべてのメモリを確保する
<code>bldmem</code>	メモリブロックをキロバイト単位で確保する
<code>calloc</code>	配列の領域を割りあてる
<code>chkml</code>	未使用のメモリブロックのうち、最大のもののサイズをバイトで通知する
<code>free</code>	<code>calloc</code> 、 <code>malloc</code> 、 <code>realloc</code> で割りあて済みのメモリブロックを解放する
<code>getmem</code>	指定されたサイズのメモリ領域を確保する
<code>getml</code>	指定されたサイズのメモリ領域を確保する
<code>malloc</code>	メモリブロックを割りあてる
<code>rbrk</code>	ブレイク値を初期状態に戻す
<code>realloc</code>	割りあて済みのメモリブロックのサイズを変更する
<code>rlsmem</code>	<code>getmem</code> で確保したメモリ領域を解放する
<code>rlsml</code>	<code>getml</code> で確保したメモリ領域を解放する
<code>rstmem</code>	ブレイク値を初期状態に戻す

3.2 メモリ割りあて

関数名	機 能
sbrk	ブレイク値を変更する（ヒープ領域を拡張する）
sizmem	使用可能なメモリブロックのサイズをワードで通知する
swab	ワードデータ列を上位／下位を入れ替えて転送する
swaw	ロングワードデータを上位／下位を入れ替えて転送する

注：sbrk 関数は他のメモリ割りあて関数と一緒に使うべきものではありません。

特に、sbrk でサイズを小さくし、他のメモリ割りあて関数を使用した場合、動作はまったく保証されません。

また、rbrk 関数も他のメモリ割りあて関数と一緒に使ってはいけません。

3.3 文字の分類と変換

これらは、文字の分類（判定）や、大文字～小文字間の変換などを行うマクロ（または関数）であり、すべてインクルードファイル `ctype.h` と `fctype.h` で宣言されています。

これらのマクロ（または関数）には以下のものがあります。

関数名	機 能
<code>isalnum</code>	英数字の判定を行う
<code>isalpha</code>	英文字の判定を行う
<code>isascii</code>	ASCII 文字の判定を行う
<code>isctrl</code>	制御文字の判定を行う
<code>isctype</code>	C 言語の識別子として使用できる文字の判定を行う
<code>isctypef</code>	C 言語の識別子の先頭文字として使用できる文字の判定を行う
<code>isdigit</code>	10 進数の判定を行う
<code>isgraph</code>	空白以外の表示可能文字の判定を行う
<code>islower</code>	英小文字の判定を行う
<code>isodigit</code>	8 進数の判定を行う
<code>isprint</code>	表示可能文字の判定を行う
<code>ispunct</code>	句読点の判定を行う
<code>isspace</code>	空白文字の判定を行う
<code>isupper</code>	英大文字の判定を行う
<code>isxdigit</code>	16 進数の判定を行う
<code>toascii</code>	与えられた文字を ASCII コードへ変換する
<code>tolower</code>	文字を判定し、大文字ならば小文字へ変換する

3.3 文字の分類と変換

関数名	機 能
<code>toupper</code>	文字を判定し、小文字ならば大文字へ変換する
<code>_tolower</code>	大文字を小文字へ変換する (A~Z 以外は不定)
<code>_toupper</code>	小文字を大文字へ変換する (a~z 以外は不定)

これらは、マクロと関数の両方が用意されています。

`ctype. h` をインクルードするとマクロが、`fctype. h` をインクルードすると関数が使用されます。

これらのうち、`isodigit`、`iscsym`、`iscsymf`、`tolower`、`toupper` を関数として使用する場合は良いのですが、マクロとして使用する際には、注意が必要となります。

以上の5つは、マクロで使用した場合に引数を複数回評価しますので、副作用を伴う引数を指定した場合のみ、結果が正しく得られない場合があります。

他の文字分類、文字変換を関数として呼び出しても良い場合には、`fctype. h` をインクルードして、すべてを関数として宣言してください。

副作用を伴わない引数を指定しているものはマクロとして、副作用を伴う引数を指定しているものは関数として宣言したい場合は、以下のようしてください。

```
#include <fctype. h> ..... 副作用のあるものを関数として宣言するため
                             fctype. h をインクルードする。
```

```
#include <ctype. h> ..... 一旦、すべてをマクロとして宣言する。
```

```
#undef tolower..... 関数として呼び出すもののみマクロの宣言を無効
                             にする。
```

この方法のように `ctype. h` と `fctype. h` の両方をインクルードする場合は、必ず `fctype. h` を先にインクルードしてください。

もし、`ctype. h` をインクルードした後で `fctype. h` をインクルードすると、`fctype. h` は無効になるので注意してください。

3.4 データ変換

数値変換関数

データ変換関数は、数値～文字列間の変換を行う関数であり、以下のような場合に使用します。

- 文字列を、コンピュータ内部で演算可能な表現形式に変換したいとき
- コンピュータ内部の数値表現形式を、画面に表示可能な、またはプリンタに印字可能な文字列に変換したいとき

データ変換関数には以下のものがあり、すべてインクルードファイル `stdlib.h` で宣言されています。

関数名	機 能
<code>atof</code>	数値を表す文字列を倍精度の浮動小数点値に変換する
<code>atoi</code>	数値を表す文字列を整数値に変換する
<code>atol</code>	数値を表す文字列を long 型整数値に変換する
<code>atow</code>	数値を表す文字列を short 型整数値に変換する
<code>ecvt</code>	倍精度の浮動小数点値を数値を表す文字列に変換する
<code>fcvt</code>	倍精度の浮動小数点値を数値を表す文字列に変換する
<code>gcvt</code>	倍精度の浮動小数点値を数値を表す文字列に変換する
<code>itoa</code>	整数を文字列に変換する
<code>ltoa</code>	long 型整数を文字列に変換する
<code>strtod</code>	数値を表す文字列を浮動小数点値に変換する
<code>strtoul</code>	符号が付いても良い数値を表す文字列を long 型整数値に変換する
<code>strtoul</code>	符号が付かない数値を表す文字列を unsigned long 型整数値に変換する
<code>uitoa</code>	符号なし整数を文字列に変換する

3.5 ディレクトリ制御

ディレクトリ制御関数は、ディレクトリに関する操作を行います。
ディレクトリ制御関数には以下のものがあり、すべてインクルードファイル
direct.hで宣言されています。

関数名	機 能
chdir	現在のワーキングディレクトリを変更する
getcwd	現在のワーキングディレクトリを通知する
mkdir	ディレクトリを作成する
rmdir	ディレクトリを削除する

3.6 ファイル操作

ファイル操作関数は、パス名やファイルハンドルで指定したファイルの管理情報の通知と変更、ファイルサイズの通知と変更、ファイル名の変更、ファイルの削除などを行います。

これらの関数には以下のものがあり、fstat、stat 関数はインクルードファイル stat. h で、これ以外の関数はインクルードファイル io. h で宣言されています。

関数名	機 能
access	ファイルの許可設定を調べる
chgfa	ファイルの属性を変更する
getfa	ファイルの属性を調べる
chgft	ファイルの作成日時を変更する
getft	ファイルの作成日時を調べる
dtasc	ファイル情報の日付から文字列を作る
dtcnv	文字列からファイル情報用の日付を作る
tmsc	ファイル情報の時刻から文字列を作る
tmcnv	文字列からファイル情報用の時刻を作る
chmod	ファイルの許可設定を変更する
chsize	ファイルの大きさを変更する
filelength	ファイルの長さを調べる
files	ファイルを検索しファイルの情報を得る
nfiles	ファイルを検索しファイルの情報を得る
fstat	ファイルハンドルのステータス情報を得る
ftpack	年月日時分秒をファイル情報用の日時にする
ftunpk	ファイル情報用の日時を年月日時分秒にする

関数名	機 能
isatty	文字デバイスであるかどうかを調べる
mktemp	仮のファイル名を作る
rename	ファイル名を変更する
setmode	ファイルの変換モードを設定する
stat	ファイル名からファイルのステータス情報を得る
unlink	ファイルを削除する

以下の関数を使用するときは、ファイルはパス名またはファイル名によって指定します。

access

chgfa

chmod

files

getfa

nfiles

rename

remove

stat

unlink

utime

以下の関数では、ファイルの指定はファイルハンドルで行います。

chgft

chsize

filelength

fstat

getft

isatty

setmode

3.7 ストリーム入出力

ストリーム入出力関数は、C 標準ライブラリの入出力ルーチンのうちの1つですが、他の入出力ルーチンと比べた場合、以下のような特徴があります。

- データファイルを独立した文字のストリームとして扱う
- データのバッファリングが可能となる
- データをフォーマット化して入出力を行うことができる
- ファイルの管理は FILE 構造体によって行われる

これらの詳細については、この後で述べる各項目別の基本事項の中でふれます。

ストリーム入出力関数には以下のものがあり、これらの関数およびこれらの関数を使用するにあたって必要とする定数、型、構造体などは、すべてインクルードファイル `stdio.h` で定義されています。

関数名	機 能
<code>clearerr</code> (<code>clrerr</code>)	ストリームのためのエラーインジケータをクリアする
<code>fclose</code>	ストリームをクローズする
<code>fcloseall</code>	オープンされているすべてのストリームをクローズする
<code>fdopen</code>	ファイルハンドルを使ってストリームをオープンする
<code>fEOF</code>	指定されたストリームがエンドオブファイルかどうか調べる (マクロ)
<code>ferror</code>	ストリームの入出力時のエラーを調べる (マクロ)
<code>fflush</code>	ストリームをフラッシュする

3.7 ストリーム入出力

関数名	機 能
fgetc	ストリームから1文字読み込む (関数)
fgetchar	標準入力から1文字読み込む (関数)
fgetpos	現在のファイルポインタの位置を得る
fsetpos	ファイルポインタを所定の位置へ移動する
fgets	ストリームから文字列を読み込む
fileno	ストリームのためのファイルハンドルを得る (マクロ)
flushall	すべてのストリームをフラッシュする
fmode	ファイルの変換モードを変更する
fopen	ストリームをオープンする
fprintf	データをフォーマットしてストリームに書き込む
fputc	ストリームへ1文字書き込む (関数)
fputchar	標準出力へ1文字書き込む (関数)
fputs	ストリームへ文字列を書き込む
fread	ストリームから固定長データ項目を読み込む
freopen	ストリームポインタを再割りつける
fscanf	ストリームからデータをフォーマットにしたがって読み込む
fseek	ファイルポインタを所定の位置へ移動する
ftell	現在のファイルポインタの位置を得る
fwrite	ストリームへ固定長データ項目を書き込む
getc	ストリームから1文字読み込む (マクロ)
getchar	標準入力から1文字読み込む (マクロ)
getl	ストリームから int 型のデータを読み込む
gets	標準入力から1行読み込む

3.7 ストリーム入出力

関数名	機 能
getw	ストリームから short 型のデータを読み込む
printf	標準出力へデータをフォーマットして書き込む
putc	ストリームへ 1 文字書き込む (マクロ)
putchar	標準出力へ 1 文字書き込む (マクロ)
putl	ストリームへ int 型のデータを書き込む
puts	ストリームへ 1 行書き込む
putw	ストリームへ short 型のデータを書き込む
remove	ファイルを削除する
rewind	ファイルポインタをファイルの先頭に移動する
rmtemp	すべての作業用ファイルを削除する
rmtmp	すべての作業用ファイルを削除する
tempfile	作業用ファイルを新規に作成する
tmpfile	作業用ファイルを新規に作成する
tempnam	作業用ファイル用のファイル名を作る
tmpnam	作業用ファイル用のファイル名を作る
scanf	標準入力からデータをフォーマットして読み込む
setbuf	ストリームバッファの制御を行う
setnbf	ストリームバッファを変更する
setvbuf	ストリームバッファを変更する
sprintf	文字列へデータをフォーマットして書き込む
sscanf	文字列からデータをフォーマットして読み込む
ungetc	読み込んだ文字を入力ストリームに返す
vfprintf	データをフォーマットしてストリームに書き込む
vprintf	標準出力へデータをフォーマットして書き込む
vsprintf	文字列へデータをフォーマットして書き込む

3.7 ストリーム入出力

注：ストリーム入出力関数と低水準入出力関数は一般に互換性がないので1個のファイルに対して両者を併用してはなりません。

たとえば、ストリーム入出力関数はバッファリングを行うのに対して、低水準入出力関数はこれを行わないので、両者を併用してアクセスすると、バッファ内のデータが失われることがあります。

getc、putc、getchar、putchar、feof、ferror、fileno、clearerr の8つの関数は、マクロと関数の両方が用意されています。

マクロを使用する場合には、stdio. hをインクルードする前に以下のようにしてください。

```
#define MACRO
```

関数を使用する場合には、stdio. hをインクルードする前に以下のようにしてください。

```
#undef MACRO
```

ストリームのオープン

ストリームの入出力を行うに先立って、ストリームをオープンしなければなりません。

ストリームのオープンは、fopen、fdopen、freopen 関数によって行います。これらの関数は、FILE 構造体と呼ばれる、ファイルの管理に必要な情報を格納している構造体へのポインタを返してきますので、以後このポインタを使ってストリームへのアクセスが行えるようになります。

定義済みのストリームポインタ

ストリームポインタの中には、自動的にオープンされるため、上記の関数によるオープン処理を必要としないものもあります。

このようなストリームポインタを以下の表に示します。

3.7 ストリーム入出力

ストリーム	対応する入出力デバイス
stdin	標準入力（キーボード）
stdout	標準出力（画面）
stderr	標準エラー出力（画面）
stdaux	標準補助入出力（RS-232C）
stdprn	標準プリンタ出力（プリンタ）

なお、標準入出力の入力先、および出力先は、入出力リダイ렉션機能により変更することができます。

ただし、入出力リダイレクトされているときに `fclose` や `fcloseall` 関数などによってクローズしてしまうと、入出力リダイレクトは無効になってしまうので注意してください。

詳細については、「C ユーザーマニュアル」を参照してください。

ストリームバッファの制御

ストリーム入出力関数によってオープンしたファイルは、自動的にバッファリング、すなわちバッファを確保し、このバッファを経由して入出力が行われるようになります。

ただし、オープン済みのストリーム `stdin` や `stdout` などはこのかぎりではありません。

これらのストリームはラインバッファ化され、改行を入力するとフラッシュされます。

`setbuf` 関数によって、以下のような、バッファに対する特殊な制御を行うことができます。

- ストリームのバッファリングを抑止する
- バッファリングされていないストリームとバッファを結合する
- (`setbuf` 関数によって割りつけられた) システムバッファをユーザーが命名し、アクセスできるようにする

バッファサイズは、インクルードファイル `stdio.h` の中で、記号定数 `BUF_SIZE` で定義されており、バッファのアクセスに際しては、このサイズを使用します。

3.7 ストリーム入出力

プログラムでストリームからの入力を行うと、データはいったんバッファへ読み込まれ、次にユーザー領域へ転送されます。

次にストリームの読み込み要求を行うと、データはファイルデバイスからではなく、このバッファから読み込まれます。

バッファからすべてのデータが読み込まれると、再びファイルデバイスからデータがバッファに読み込まれます。

これに対して、ストリームの出力を行うと、データは次々に出力バッファへ書き込まれます。

そして出力バッファがフルになると、自動的にファイルデバイスへの書き出しが行われます。

また、出力バッファがフルになっていなくても、ファイルがクローズされると、自動的に書き出しが行われます。

これを、バッファのフラッシュといいます。

上記で述べたように、通常、バッファのフラッシュは自動的に行われますが、`fflush`、`flushall` 関数により、強制的にフラッシュすることもできます。

ファイルポインタによるストリームのアクセス

ストリームへのアクセス、すなわちストリームのデータの読み込み、およびストリームへのデータの書き込みは、ファイルポインタにより行います。

ファイルポインタは、通常、システムによって設定と更新が行われます。たとえば、

- ストリームを読み込みモードでオープンしたときには、ファイルポインタはファイルの先頭に設定され、1文字読み込むたびに、次に読み込むべき位置を指すように更新される。
- ストリームを書き込みモードでオープンしたときには、ファイルポインタはファイルの先頭に設定され、1文字書き込むたびに、次に書き込むべき位置を指すように更新される。
- ストリームを追加（アペンド）モードでオープンしたときには、ファイルポインタはファイル内の最終データの次の位置に決定され、1文字書き込むたびに、次に書き込むべき位置を指すように更新される。

これに対して、複雑なファイル処理が必要となり、ユーザー自身がファイルポインタを操作するために、以下の関数が用意されています。

3.7 ストリーム入出力

fseek ……ファイルポインタを任意の位置に設定する
rewind ……ファイルポインタをファイルの先頭に設定する
ftell ……ファイルポインタの現在の位置を通知する
feof ……ファイルポインタの現在の位置がエンドオブファイルである
か否か通知する

注：fseek 関数を使用する場合、テキストモードでオープンしたファイルの
ファイルポインタを変更する場合には、以下のいずれかの条件に合わない
位置に変更してはいけません。

- ファイルの先頭の位置
- ftell 関数で返された位置
- ファイルの最終データの次の位置

エラーの発見

ストリームの入出力によってエラーが発生した場合、ストリーム上のエラー
インジケータがセットされます。
エラーが発生したか否かは、ferror 関数により判定することができます。

エラーインジケータがセットされると、以下の処理を行わないかぎりエラー
インジケータは現在の状態を保存します。

- ストリームをクローズする
- rewind 関数により、ファイルをリワインドする
- clearerr 関数により、エラーインジケータをクリアする

ストリームのクローズ

ファイルの処理が終了したら、fclose、fcloseall 関数により、ファイルをク
ローズします。

fcloseall 関数は、現在処理中のファイルだけでなく、オープンされている
ファイルをすべてクローズします。

なお、ストリーム stdin、stdout などはクローズする必要はありません。
プログラムが終了すれば、オープン済みのファイルはすべてクローズされま
すが、オープンできるファイルの数に限りがある場合もありますので、オー
プンしたファイルは必ずクローズするようにしましょう。

3.8 低水準入出力

低水準入出力 8.8

低水準入出力関数は、オペレーティングシステムの入出力機能を直接呼び出すもので、他の入出力ルーチンと比較した場合、以下のような特徴があります。

- データのバッファリングとフォーマット化は行われない
- オープンされたファイルは、ファイルハンドルによって参照される

低水準入出力関数には以下のものがあり、これらの関数はすべてインクルードファイル `io.h` で宣言されています。

関数名	機 能
<code>close</code>	ファイルをクローズする
<code>creat</code>	ファイルを新規に作成する
<code>dup</code>	ファイルハンドルを複製する
<code>dup2</code>	ファイルハンドルを強制的に再割りつけする
<code>eof</code>	ファイルがエンドオブファイルであるか否か判定する
<code>lseek</code>	ファイルポインタを指定位置へ移動する
<code>open</code>	ファイルをオープンする
<code>read</code>	ファイルからデータを読み込む
<code>tell</code>	現在のファイルポインタの位置を得る
<code>write</code>	ファイルにデータを書き込む

ファイルのオープン

ファイルの入出力を行うに先立って、ファイルをオープンしておかなければなりません。

ファイルのオープンには、`open`、`creat` 関数によって行います。

これらの関数は、ファイルハンドル（ファイル記述子）を返してきますので、以後これを使ってファイルをアクセスすることができます。

ファイルは読み込み、書き込み、あるいはその両方のモードで、またテキストモードでもバイナリモードでもオープンすることができます。

定義済みのファイルハンドル

以下の表にある5つのファイルハンドルは自動的にオープンされるので、ファイルをオープンすることなく、低水準入出力関数から使用することができます。

また、ストリーム入出力関数と同様に、標準入出力のリダイレクションを行うことができます。

ハンドル	対応するストリーム	対応する入出力デバイス
0	stdin	標準入力（通常はキーボード）
1	stdout	標準出力（通常はスクリーン）
2	stderr	標準エラー出力（スクリーン）
3	stdaux	標準補助入出力（RS-232C）
4	stdprn	標準プリンタ出力（プリンタ）

データの読み出し／書き込み

データの入出力は、read および write という2つの基本的な関数により行います。

ファイルの読み出しと書き込みは、ストリーム入出力関数と同様に、ファイルポインタによって行います。

ファイルポインタは読み書きのつど、自動的に更新されますが、以下の関数により、ファイルポインタの操作を行うことができます。

lseek	………	ファイルポインタを任意の位置に設定する
tell	………	ファイルポインタの現在の位置を通知する
eof	………	ファイルポインタの現在の位置がエンドオブファイルである か否か通知する

注：lseek 関数を使用する場合、テキストモードでオープンしたファイルのファイルポインタを変更する場合には、以下のいずれかの条件に合わない位置に変更してはいけません。

- ファイルの先頭の位置
- tell 関数で返された位置
- ファイルの最終データの次の位置

ファイルのクローズ

ファイルのクローズは、close 関数を使用して行います。

プログラムが終了すれば、オープンされていたファイルはすべて自動的にクローズされますが、オープンできるファイルの数が限られている場合もあるので、オープンしたファイルは必ずクローズするようにしましょう。

関 数	説明
close	ファイルを開いたときに指定したモードで開いたファイルをクローズする。
fclose	標準入出力ファイル（stdin, stdout, stderr）をクローズする。
fcloseall	すべての標準入出力ファイルをクローズする。
fflush	バッファをフラッシュ（出力）する。
fgetc	ファイルから1バイトを読み取る。
fputc	1バイトをファイルに書き込む。
fgetchar	標準入力（stdin）から1バイトを読み取る。
f_putchar	1バイトを標準出力（stdout）に書き込む。
fread	バイナリ形式でデータをファイルから読み取る。
fwrite	バイナリ形式でデータをファイルに書き込む。
rewind	ファイルのポインタを最初に戻す。
seek	ファイルのポインタを指定した位置に移動する。
setbuf	標準入出力のバッファを設定する。
setvbuf	ファイルのバッファを設定する。
ungetc	読み取った1バイトをファイルのポインタに戻す。
vfprintf	フォーマット指定子に従ってデータをファイルに書き込む。
vprintf	標準出力（stdout）にフォーマット指定子に従ってデータを出力する。

3.9 コンソール入出力

これらの関数は、コンソールと直接入出力を行うものですが、コマンド行で入出力のリダイレクションを行うことも可能です。

コンソール入出力関数には以下のものがあり、これらはすべてインクルードファイル `conio.h` で宣言されています。

関数名	機 能
<code>cgets</code>	コンソールから文字列を読み込む
<code>cprintf</code>	コンソールヘデータをフォーマットして出力する
<code>cputs</code>	コンソールへ文字列を出力する
<code>cscanf</code>	コンソールからデータをフォーマットして読み込む
<code>getch</code>	コンソールから 1 文字読み込む
<code>getche</code>	コンソールから 1 文字読み込み、エコーバックする
<code>kbhit</code>	キー入力の有無を調べる
<code>putch</code>	コンソールへ 1 文字出力する
<code>ungetch</code>	コンソールから読み込んだ文字を、次の読み込みのために返す
<code>vcprintf</code>	コンソールヘデータをフォーマットして出力する

3.10 数値演算

数値演算 51

これらの関数は、一般的な数値演算を行います。

数値演算関数では、浮動小数点を扱うので、浮動小数点サポートが必要となります。

数値演算関数には以下のものがあり、div、ldiv 関数がインクルードファイル `stdlib.h` で宣言されているほかは、すべて `math.h` で宣言されています。

関数名	機 能
<code>acos</code>	アークコサインを求める
<code>asin</code>	アークサインを求める
<code>atan</code>	アークタンジェントを求める
<code>atan2</code>	アークタンジェントを求める
<code>cabs</code>	複素数の絶対値を求める
<code>ceil</code>	Xと等しいかXより大きい最小の整数値を求める
<code>cos</code>	コサインを求める
<code>cosh</code>	ハイパボリックコサインを求める
<code>div</code>	int の割り算の商と余りを得る
<code>except</code>	数値演算ライブラリ使用時の例外処理を行う
<code>exp</code>	指数関数を求める
<code>fabs</code>	浮動小数点値の絶対値を求める
<code>floor</code>	Xと等しいかXより小さい最大の整数値を求める
<code>fmod</code>	剰余を求める
<code>frexp</code>	浮動小数点値を仮数部と指数部に分ける
<code>hypot</code>	直角三角形の斜辺の長さを求める
<code>ldexp</code>	X と 2 の n 乗の積を求める

3.10 数値演算

関数名	機 能
ldiv	long の割り算の商と余りを得る
log	自然対数を求める
log10	10 を底とした対数を求める
matherr	数値演算関数のエラー処理を行う
modf	浮動小数点値を整数部と分数部に分ける
pow	X^Y を求める
sin	サインを求める
sinh	ハイパボリックサインを求める
sqrt	平方根を求める
tan	タンジェントを求める
tanh	ハイパボリックタンジェントを求める

3.11 プロセス制御

時間とメモリ

これらの関数はプロセスの制御を行います。

「プロセス」とは、オペレーティングシステムにおける実行の単位であり、プログラムのコード、データ、資源、およびこれらに関する制御情報などから成ります。

プロセス制御関数は、具体的には、以下のことを行います。

- プロセスを終了させる
- シグナル割り込みの制御を行う
- 新しいプロセスを起動する

プロセス制御関数には以下のものがあり、signal、raise 関数はインクルードファイル signal.h で宣言されています。

また、abort、atexit、exit、_exit、system の各関数は、インクルードファイル stdlib.h で宣言されています。

これ以外の関数はインクルードファイル process.h で宣言されています。

関数名	機 能
abort	プロセスを異常終了する
atexit	exit 時に呼び出す関数を登録する
execl	引数リストによって子プロセスを実行する
execle	引数リストと与えられた環境により子プロセスを実行する
execlp	引数リストと path 変数によって子プロセスを実行する
execlpe	引数リスト、path 変数、および引数 envp により子プロセスを実行する
execv	引数配列によって子プロセスを実行する

3.11 プロセス制御

関数名	機 能
execve	引数配列と与えられた環境によって子プロセスを実行する
execvp	引数配列と path 変数によって子プロセスを実行する
execvpe	引数配列、path 変数、および引数 envp により子プロセスを実行する
exit	プロセスを終了する
_exit	バッファをフラッシュせずに子プロセスを終了する
getpid	プロセスの ID を得る
raise	プログラムにシグナルを送る
signal	シグナル割り込みのための操作を行う
spawnl	引数リストによって子プロセスを実行する
spawnle	引数リストと与えられた環境により子プロセスを実行する
spawnlp	引数リストと path 変数によって子プロセスを実行する
spawnv	引数配列によって子プロセスを実行する
spawnve	引数配列と与えられた環境によって子プロセスを実行する
spawnvp	引数配列と path 変数によって子プロセスを実行する
system	Human68k コマンドを実行する
wait	起動した子プロセスの終了コードを得る

これらの関数では、子プロセスを起動するための path 変数の設定の有無、子プロセスへの引数のわたしかた、子プロセスの実行環境（の設定方法）の違いにより、8つのパターンに分けられるので、これを次頁の表に示します。

関数名	path 変数の設定	引数の わたしかた	子プロセスの実行環境
execl spawnl	path は使用しない	引数リスト	親プロセスの環境を 引き継ぐ
execle spawnle	path は使用しない	引数リスト	最後の引数として子 プロセスの環境テー ブルがわたされる
execlp spawnlp	path を使用する	引数リスト	親プロセスの環境を 引き継ぐ
execv spawnv	path は使用しない	引数配列	親プロセスの環境を 引き継ぐ
execve spawnve	path は使用しない	引数配列	最後の引数として子 プロセスの環境テー ブルがわたされる
execvp spawnvp	path を使用する	引数配列	親プロセスの環境を 引き継ぐ
execlpe	path を使用する	引数リスト	最後の引数として子 プロセスの環境テー ブルがわたされる
execvpe	path を使用する	引数配列	最後の引数として子 プロセスの環境テー ブルがわたされる

3.12 サーチとソート

これらの関数は、インクルードファイル `stdlib.h` で宣言されています。

関数名	機 能
<code>bsearch</code>	バイナリサーチを実行する
<code>dqsort</code>	double 型のデータをソートする
<code>fqsort</code>	float 型のデータをソートする
<code>lqsort</code>	long 型のデータをソートする
<code>qsort</code>	クイックソートを実行する
<code>qsqsort</code>	short 型のデータをソートする
<code>tqsort</code>	ポインタ型のデータをソートする

3.13 文字列操作

各関数の説明

これらの関数は、ヌル文字で終了する文字列の操作を行います。

ヌル文字で終了しない文字列については、バッファ操作関数を使用してください。

文字列操作関数には、以下のものがあり、これらはすべてインクルードファイル `string.h` で宣言されています。

関数名	機能
<code>strcat</code>	文字列を追加する
<code>strchr</code>	文字列において、指定された文字の最初の位置を調べる
<code>strcmp</code>	2つの文字列を比較する
<code>strcmpi</code>	2つの文字列を大文字／小文字の区別なしで比較する
<code>strcpy</code>	文字列を指定された場所へコピーする
<code>strcspn</code>	文字列において、指定文字セットの最初の文字を探す
<code>strdup</code>	文字列を複写する
<code>stricmp</code>	2つの文字列を大文字／小文字の区別なしで比較する
<code>strlen</code>	文字列の長さを調べる
<code>strlwr</code>	文字列中の大文字をすべて小文字に変換する
<code>strncat</code>	文字列の先頭から n 文字分、他の文字列に追加する
<code>strncmp</code>	先頭から n 文字分、2つの文字列を比較する
<code>strncpy</code>	文字列中の n 文字を他の文字列にコピーする

3.13 文字列操作

関数名	機 能
strnset	文字列の先頭 n 文字を、指定された文字で初期化する
strpbrk	他の文字列の構成文字を、指定文字列中で探す
strrchr	文字列中で、指定された文字が最後に現れる位置を探す
strrev	文字列の文字の並びを逆にする
strset	文字列の全文字を、指定された文字で置き換える
strsfn	ファイル名をドライブ名、パス名、主ファイル名、拡張子に分解する
strspn	文字列中から、指定文字セット以外の文字の位置を探す
strstr	文字列の中から指定した文字列と一致する部分を検索する
strtok	文字列の次のトークンを探す
strupr	文字列中の小文字をすべて大文字に変換する
strins	文字列を、他の文字列の前に挿入する
strmfe	ファイル名の拡張子を置き換える
strmfn	ドライブ名、パス名、主ファイル名、拡張子からファイル名を作成する
strmfp	パス名、主ファイル名からファイル名を作成する
stcgfe	ファイル名から拡張子を取り出す
strbpl	文字列の並びをもとに、ポインタ配列を作る
stcgfn	ファイル名から主ファイル名を取り出す
strsrt	文字列の配列を ASCII コード順にソートする

3.14 時間

関数 3.14

時間関数は、システム時間から現在の時間を得るとともに、必要に応じてそれを変換し、格納します。

たとえば、time、ftime 関数は、グリニッジ標準時の 1970 年 1 月 1 日から現在まで経過した時間を秒単位で返します。

また、上記の関数で得た時間は、asctime、ctime、gmtime、localtime 関数を使用することにより、さまざまな型に変換することができます。

時間関数で使用するインクルードファイルは以下の通りです。

- ftime、utime 以外 …… time. h (関数宣言)
- ftime …… timeb. h (関数宣言)
- utime …… utime. h (関数宣言)

ftime、localtime の地方時調整には、グローバル変数 daylight、timezone が参照されます。

これらは、日本での値で初期化されています。

上記の変数については、「第2章 2.3 グローバル変数」の daylight、timezone をそれぞれ参照してください。

関数名	機 能
asctime	tm 構造体の時間情報を文字列に変換する
ctime	時間を time_t 型数値から文字列に変換する
ftime	現在のシステム時間を tm 構造体で得る
gmtime	時間を time_t 型数値から tm 構造体に変換する
time	現在のシステム時間を time_t 型数値で得る
tzset	環境時間変数から外部時間変数を設定する
clock	プログラムが起動してからの時間を得る

3.14 時間

関数名	機 能
difftime	時間の差を浮動小数点値で得る
localtime	現在の地方時を得る
mktime	tm 構造体から time_t 型数値に変換する
strftime	tm 構造体で指定する日時を指定した書式で文字列に変換する
utime	ファイルの作成日時を変更する

関 名	機 能
asctime	tm 構造体の時間情報を文字列に変換する
ctime	時間を time_t 型数値から文字列に変換する
time	現在のシステム時間を tm 構造体で得る
gmtime	時間を time_t 型数値から tm 構造体に変換する
time	現在のシステム時間を time_t 型数値で得る
tzset	標準時間変数から外部時間変数を設定する
clock	プロセスが経過した時間の得る

3.15 日本語ライブラリ

日本語ライブラリには、日本語処理において入力される文字の分類を行う関数と、漢字処理関数があります。

3.15.1 文字分類関数

文字分類関数には以下のものがあり、すべてインクルードファイル `jfctype.h` で宣言されています。

関数名	機 能
<code>iskana</code>	カナ文字またはカナ句読点の判定を行う
<code>iskpun</code>	カナ句読点の判定を行う
<code>iskmoji</code>	カナ文字の判定を行う
<code>isalkana</code>	英文字またはカナ文字の判定を行う
<code>ispnkana</code>	英句読点またはカナ句読点の判定を行う
<code>isalnmkana</code>	英数字またはカナ文字の判定を行う
<code>isprkana</code>	表示可能文字（空白を含む）の判定を行う
<code>isgrkana</code>	表示可能文字（空白を除く）の判定を行う
<code>iskanji</code>	漢字の1バイト目の判定を行う
<code>iskanji2</code>	漢字の2バイト目の判定を行う

3.15.2 漢字処理関数

漢字処理関数には、漢字を含む文字列の操作を行う関数と、全角文字の分類、および全角文字～半角文字の変換を行う関数の2種類があります。

漢字を含む文字列の操作を行う関数には、以下のものがあります。

3.15 日本語ライブラリ

これらはすべて、インクルードファイル `jstring.h` で宣言されています。

関数名	機 能
<code>jstrncmp</code>	<code>strncmp</code> 関数の漢字対応版で、漢字を含む2つの文字列を先頭から <code>n</code> 文字分比較する
<code>jstrchr</code>	<code>strchr</code> 関数の漢字対応版で、漢字を含む文字列において、指定文字が最初に現れる位置を調べる
<code>jstrrchr</code>	<code>strrchr</code> 関数の漢字対応版で、漢字を含む文字列において、指定文字が最後に現れる位置を調べる
<code>jstrcmp</code>	<code>strcmp</code> 関数の漢字対応版で、漢字を含む2つの文字列の比較を行う

全角文字の分類、および全角文字～半角文字の変換を行う関数には以下のものがあります。

これらはすべて、インクルードファイル `jfctype.h` で宣言されています。

関数名	機 能
<code>jiszen</code>	全角文字の判定を行う
<code>jisl0</code>	全角空白文字または JIS 第一水準の記号の判定を行う
<code>jisl1</code>	JIS 第一水準漢字の判定を行う
<code>jisl2</code>	JIS 第二水準漢字の判定を行う
<code>jisalpha</code>	全角英文字の判定を行う
<code>jisupper</code>	全角英大文字の判定を行う
<code>jislower</code>	全角英小文字の判定を行う
<code>jisdigit</code>	全角数字の判定を行う
<code>jiskata</code>	全角カタカナ文字の判定を行う

3.15 日本語ライブラリ

関数名	機 能
jishira	全角ひらがな文字の判定を行う
jiskigou	全角句読点文字の判定を行う
jisspace	全角空白文字の判定を行う
hantozen	半角文字を全角文字に変換
zentohan	全角文字を半角文字に変換

関 名	機 能
a_dplay	PCM音声を再生する
a_fec	PCM音声を自己修正する
a_com	一時中断してPCMの動作を再開する
a_end	PCMの動作を強制終了する
a_start	PCMの動作状態を初期化する
a_stop	PCMの動作を一時中断する
asc	文字列の先頭の文字コードを返す
b_bin	整数を2進数の文字列に変換する
b_chr	文字コードを文字に変換する
fix	浮動小数点数値の整数部分を返す
b_hex2	整数を16進数の文字列に変換する
b_iios	整数を文字列に変換する
b_oct2	整数を8進数の文字列に変換する
val	文字列を整数に変換する
sgn	浮動小数点数の符号を返す
dekt	指定したマスの空容量を通知する
b_fclose	指定されたファイルを閉じる

3.16 BASICライブラリ

BASIC ライブラリは、BASTOC、すなわち BASIC から C への変換処理において使用します。

BASIC ライブラリには以下のものがあり、これらはすべてインクルードファイル audio. h、basic. h、または basic0. h、graph. h、image. h、mouse. h、music. h、または music2. h、sprite. h、stick. h で宣言されています。

関数名	機 能
a_play	PCM 録音を再生する
a_rec	PCM 録音を行う
a_cont	一時中断していた PCM の動作を再開する
a_end	PCM の動作を強制終了する
a_stat	PCM の動作状態を得る
a_stop	PCM の動作を一時中断する
asc	文字列の先頭の文字コードを返す
b_binS	整数を 2 進数の文字列に変換する
b_chrS	文字コードを文字に変換する
fix	浮動小数点数値の整数部を返す
b_hexS	整数を 16 進数の文字列に変換する
b_itoa	整数を文字列に変換する
b_octS	整数を 8 進数の文字列に変換する
val	文字列を数値に変換する
sgn	浮動小数点数値の符号を得る
dskf	指定ドライブのディスクの空き容量を通知する
b_fclose	指定されたファイルをクローズする

3.16 BASICライブラリ

関数名	機 能
b_fcloseall	すべてのファイルをクローズする
b_feof	指定されたファイルがエンドオブファイルか否か調べる
b_fgetc	指定されたファイルから1文字読み込む
b_fopen	指定されたファイルをオープンする
b_fputc	指定されたファイルへ1文字書き出す
b_fread	指定されたファイルからデータを読み込む
b_freads	指定されたファイルからデータを読み込む
b_fseek	ファイルポインタの位置を移動する
b_fwrite	指定されたファイルにデータを書き出す
b_fwrites	指定されたファイルにデータを書き出す
randomize	rnd の初期化を行う
b_int	浮動小数点を整数に変換する
rnd	浮動小数点数値の擬似乱数を発生させる
instr	文字列の中から特定の文字の並びを探し出す
b_leftS	文字列を切り出す
b_midS	文字列を切り出す
b_mirrorS	文字列の並びを反転する
b_rightS	文字列を切り出す
b_strchr	文字列中の最初にみつかった指定文字位置を返す
b_stringS	文字列中の1文字を複数個並べた文字列を返す
b_strrchr	文字列中の最後にみつかった指定文字位置を返す
b_strtok	指定文字を文字列から探す
b_dateS	日付を文字列に変換して通知する
b_timeS	時刻を文字列に変換して通知する

3.16 BASICライブラリ

関数名	機 能
b_dayS	曜日を通知する
b_free	フリーエリアの大きさを調べる
b_inkeyS	キーの入力を待って読みとった1文字を返す
b_inkey0	キーの入力待ちをしないで読みとった1文字を返す
b_striS	整数値を文字列に変換する
b_strfS	浮動小数点数値を文字列に変換する
csrlin	カーソル位置の Y 座標を通知する
pos	カーソル位置の X 座標を通知する
b_spaceS	スペースで埋まった文字列を作る
using	書式を指定して数値などを文字列に変換する
beep	ベルを鳴らす
cls	テキスト画面をクリアする
b_exit	プログラムを終了して親プロセスに戻る
child	子プロセスを起動する
width	画面の1行の表示文字数を設定する
color	テキスト画面の文字属性を設定する
console	スクロール行の範囲を設定する
b_input	キーから1行入力し、書式に従って変換し変数に代入する
b_linput	文字列を入力する
locate	カーソルを移動する
b_sprint	文字列をテキスト画面に表示する
b_iprint	数値をテキスト画面に表示する
b_fprint	数値をテキスト画面に表示する

3.16 BASICライブラリ

関数名	機 能
b_tprint	テキスト画面上でタブスキップを行う
b_slprint	文字列をプリンタに出力する
b_ilprint	数値をプリンタに出力する
b_flprint	数値をプリンタに出力する
b_tlprint	タブスキップをプリンタに出力する
key	ファンクションキーに文字列をセットする
screen	表示画面のサイズ、色モード、解像度などを設定する
pi	円周率 π を返す
b_pi	円周率 π の倍数を返す
b_setdate	日付を設定する
b_settime	時刻を設定する
b_csw	カーソルを ON/OFF する
b_tpalet	テキストのパレット色を設定する
b_strncpy	文字列をコピーする
b_strcmp	文字列を比較する
b_stradd	2つの文字列の連結を行う
b_init	BASIC の初期化を実行する
frename	ファイル名を変更する
fdelete	ファイルを削除する
apage	グラフィックのアクティブページを指定する
box	ボックスを設定する
circle	円弧を描く
contrast	テキスト画面のコントラストを設定する

3.16 BASICライブラリ

関数名	機 能
fill	ボックスを描きながらパレット色でその中を塗りつぶす
get	グラフィック画面の画像データを配列にとり込む
home	グラフィック画面の表示位置を設定する
hsv	色相、濃さ、明るさに応じた色コードを返す
line	線を引く
paint	指定された領域をパレット色で塗りつぶす
palet	パレットに色コードを割りあてる
point	グラフィック画面の指定座標のパレットコードを通知する
pset	グラフィック画面上に点を描く
put	グラフィック画面に画像データを表示する
rgb	赤、緑、青に応じた色コードを返す
symbol	グラフィック画面上に文字列を表示する
vpage	グラフィック画面の各ページの表示制御を行う
window	グラフィック画面のクリッピング領域の座標を設定する
wipe	グラフィック画面をクリアする
img_set	画像とり込みの設定を行う
img_still	画像とり込みの設定を行う
img_pos	画像とり込み時の画面水平位置を補正する
img_color	とり込み画像の色調補正を行う
v_cut	スーパーインポーズ時のテレビ画面の表示を制御する
crt	テレビ、ビデオ画面の切り替えを行う

3.16 BASICライブラリ

関数名	機 能
img_ht	グラフィック画面とテレビ画面の半透明制御を行う
img_save	グラフィック画面のデータを指定ファイルに書き出す
img_load	グラフィック画面のデータを指定ファイルから読み込む
img_scrn	グラフィック画面のモード設定を行う
img_put	配列に格納されたデータをグラフィック画面に表示する
keysns	キー入力の有無を判定し、値を返す
img_home	グラフィック画面の表示位置を移動する
mouse	マウスの制御を行う
msarea	マウスカーソルの移動範囲を設定する
msbtn	マウスボタンが指定した状態になるまでの時間を返す
mspos	マウスカーソルの座標を変数に格納する
msstat	マウスの状態を変数に格納する
setmspos	マウスカーソルの位置を設定する
m_alloc	トラックバッファを確保し、それをクリアする
m_assign	FM音源のチャンネルにトラックを割りあてる
m_cont	一時中断していた演奏を再開する
m_free	指定されたトラックの残り容量を通知する
m_init	FM音源を初期化する
m_play	指定したチャンネルに割りあてたトラックデータを演奏する
m_stat	チャンネルの状態を通知する

3.16 BASICライブラリ

関数名	機 能
m_stop	指定したチャンネルの演奏を一時中断する
m_tempo	テンポを指定する
m_trk	MML による楽譜データをトラックに書き込む
m_vget	音色データを配列に読み込む
m_vset	音色データを指定した音色番号のバッファに登録する
md_cont	一時中断していた演奏を再開する (MIDI 対応)
md_init	MIDI コントローラを初期化する (MIDI 専用)
md_off	MIDI チャンネルの出力の停止 (MIDI 専用)
md_on	MIDI チャンネルの出力の開始 (MIDI 専用)
md_play	指定したチャンネルに割りあてたトラックデータを演奏する (MIDI 対応)
md_regr	MIDI コントローラのレジスタからデータを読む (MIDI 専用)
md_regw	MIDI コントローラのレジスタにデータを書く (MIDI 専用)
md_stat	MIDI チャンネルの出力の禁止許可状態を得る (MIDI 専用)
md_stop	指定したチャンネルの演奏を一時中断する (MIDI 対応)
md_wrt	MIDI にデータを出力する (MIDI 専用)
sp_clr	スプライトをクリアする
sp_color	スプライトのパレット色を設定する
sp_def	スプライトのパターンデータを定義する
sp_disp	スプライト面の表示制御を行う
sp_move	スプライトプレーンの表示位置を設定する

3.16 BASICライブラリ

関数名	機 能
sp_off	スプライトの表示を OFF にする
sp_on	スプライトの表示を ON にする
sp_pat	スプライトパターンデータを配列に読み込む
sp_init	スプライトパターンを初期化する
sp_set	スプライトプレーンデータをセットする
sp_stat	指定されたスプライトの状態を通知する
bg_scroll	バックグラウンドのスクロール座標を指定する
bg_set	バックグラウンドの設定を行う
bg_stat	指定したバックグラウンドの状態を読み出す
bg_fill	バックグラウンドテキストをデータで埋めつくす
bg_put	バックグラウンドテキストに書き込む
bg_get	バックグラウンドテキストを読み出す
stick	ジョイスティックの状態を調べる
strig	指定したジョイスティックのボタンの状態を返す

機 能	関数名
レディステータスを通知する「IOCS」コール	TRAPIS
「キー」の読み出し	B_KEYINP
「キー」の読み出しと「キー」の読み出し	B_KEYSNS
「キー」の読み出しと「キー」の読み出し	B_SFISNS
「キー」の読み出しと「キー」の読み出し	BITSNS

3.17 IOCSコール

IOCS コールは、ROM に組み込まれた IOCS ルーチンと呼び出すシステムコールです。

IOCS コールのおもな機能は以下の通りです。

- キーボード制御
- TV の表示制御
- テキストの表示制御
- RS-232C 制御
- ジョイスティック制御
- プリンタ出力制御
- ディスクの入出力制御
- 日付、アラーム制御
- ADPCM 制御
- FM 音源制御
- 割り込み制御
- マウス制御
- DMA 転送制御
- 日本語処理
- グラフィック制御
- スプライト制御

IOCS コールの関数には以下のものがあり、これらはすべてインクルードファイル `iocslib. h` で宣言されています。

関数名	機 能
TRAP15	レジスタを直接指定し IOCS コールを行う
B_KEYINP	キーデータの読み出しを行う
B_KEYSNS	キーデータバッファ状態のセンスを行う
B_SFTSNS	シフトキーの押下状態を調べる
BITSNS	キーの押下状態を調べる

関数名	機 能
SKEYSET	指定キーを押したことにする
TVCTRL	TV コントロールコードを指定し、TV の操作を行う
LEDMOD	LED キーのモードを設定する
TGUSEMD	画面のモードを制御する
DEFCHR	指定の漢字パターンを外字エリアに書き込む
CRTMOD	CRT のモードを設定する
CONTRAST	コントラストを設定する
HSVTORGB	色相、濃さ、明るさに応じた色コードを作る
TPALET	テキストのパレットを指定する
TPALET2	テキストのパレットを指定する
TCOLOR	テキストのカラーを指定する
FNTGET	指定の漢字パターンを指定アドレスへ読み込む
TEXTGET	指定のドット座標からパターンを指定アドレスへ読み込む
TEXTPUT	指定アドレスからパターンを指定のドット座標へ書き込む
CLIPPUT	指定したクリップ座標に従いクリッピング処理を行う
SCROLL	テキスト／グラフィックの表示座標の設定を行う
B_CURON	カーソルを ON にする
B_CUROFF	カーソルを OFF にする
B_PUTC	指定した1バイトデータを表示する
B_PRINT	指定した文字列データを表示する
B_COLOR	カラー属性を設定する
B_LOCATE	カーソルを指定位置に設定する

3.17 IOCSコール

関数名	機 能
B_DOWN_S	1 行下へカーソル移動する (最下行のときスクロール UP)
B_UP_S	1 行上へカーソル移動する (最上行のときスクロール down)
B_UP	n 行上へカーソル移動する
B_DOWN	n 行下へカーソル移動する
B_RIGHT	n 桁右へカーソル移動する
B_LEFT	n 桁左へカーソル移動する
B_CLR_ED	カーソル位置から最終行右端まで画面をクリアする
B_CLR_ST	先頭行左端からカーソル位置まで画面をクリアする
B_CLR_AL	画面全体をクリアする
B_ERA_ED	カーソル位置から行の右端まで現在行をクリアする
B_ERA_ST	左端からカーソル位置まで現在行をクリアする
B_ERA_AL	現在行全体をクリアする
B_INS	現在カーソルが設定されている行に n 行挿入する
B_DEL	現在カーソルが設定されている行から n 行削除する
B_CONSOL	表示範囲を指定する
B_PUTMES	カーソル位置、カラー属性、表示文字数を指定して文字列を表示する
SET232C	RS-232C のモードを設定する
LOF232C	RS-232C の受信バッファ内のデータ数を求める
INP232C	RS-232C の受信データを読み込む

3.17 IOCSコール

関数名	機 能
ISNS232C	RS-232C の受信データをチェックする
OSNS232C	RS-232C への送信が可能か否かチェックする
OUT232C	RS-232C への送信を行う
JOYGET	ジョイスティックのデータを読み込む
INIT_PRN	プリンタポートを初期化する
SNSPRN	プリンタ出力が可能か否かチェックする
OUTLPT	プリンタ出力を行う (データ直接)
OUTPRN	プリンタ出力を行う (シフト JIS で漢字処理あり)
B_SEEK	指定トラック (レコード) までシークする
B_VERIFY	データの比較チェックを行う
B_READDI	診断のためのチェックを行う (2HD のみ)
B_DSKINI	ディスクの初期化を行う
B_DRVSNS	ディスクのステータスを調べる
B_WRITE	ディスクにデータを書き込む
B_READ	ディスクからデータを読み込む
B_RECALI	トラック 0 へシークを行う
B_WRITED	削除データを書き込む (2HD のみ)
B_BADFMT	不良トラックを使用不可能にする
B_READDL	削除データを読み込む (2HD のみ)
B_FORMAT	物理フォーマットを行う
B_DRVCHK	指定ドライブの状態チェックと状態設定を行う
B_EJECT	イジェクトを行う
BINDATEBCD	日付を時計の形式に変換する

3.17 IOCSコール

関数名	機 能
BINDATESET	日付を時計に設定する
TIMEBCD	時刻を時計の形式に変換する
TIMESET	時刻を時計に設定する
BINDATEGET	時計から日付を読み込む
DATEBIN	日付を BCD 表現から 2 進表現に変換する
TIMEGET	時計から時刻を読み込む
TIMEBIN	時刻を BCD 表現から 2 進表現に変換する
DATECNV	日付を表す文字列を 2 進データに変換する
TIMECNV	時刻を表す文字列を 2 進データに変換する
DATEASC	2 進データを、日付を表す文字列に変換する
TIMEASC	2 進データを、時刻を表す文字列に変換する
DAYASC	2 進データを、曜日を表す文字列に変換する
ALARMMOD	アラームの禁止／許可を設定する
ALARMSET	アラームの時刻と処理アドレスを設定する
ALARMGET	アラームの時刻と処理アドレスを読み込む
ADPCMOUT	ADPCM ヘデータを出力する
ADPCMINP	ADPCM からデータを入力する
ADPCMAOT	ADPCM ヘデータを出力する (アレイチェーンモード)
ADPCMAIN	ADPCM からデータを入力する (アレイチェーンモード)
ADPCMLOT	ADPCM ヘデータを出力する (リンクアレイチェーンモード)
ADPCMLIN	ADPCM からデータを入力する (リンクアレイチェーンモード)
ADPCMSNS	ADPCM の実行モードをセンスする

3.17 IOCSコール

関数名	機 能
ADPCMMOD	ADPCM の実行を制御する
OPMSET	FM 音源 (YM2151) にデータを書き込む
OPMSNS	FM 音源 (YM2151) のステータスを読み込む
OPMINTST	FM 音源 IC による割り込みを制御する
TIMERDST	MFP の TIMER-D による割り込みを制御する
VDISPST	V-DISP のカウンタの割り込みを制御する
CRTCRAST	CRTC に設定した割り込みラスタによる割り込みを制御する
HSYNCST	H-SYNC の起動による割り込みを制御する
PRNINTST	プリンタの割り込みを制御する
MS_INIT	マウスの初期化を行う
MS_CURON	マウスカーソルを表示する
MS_CUROF	マウスカーソルを消す
MS_STAT	マウスカーソルの表示モードを調べる
MS_GETDT	マウスの移動量とボタンの ON/OFF を調べる
MS_CURGT	マウスカーソルの座標を調べる
MS_CURST	マウスカーソルの座標を設定する
MS_LIMIT	マウスカーソルの移動範囲を設定する
MS_OFFTM	マウスボタンを離すまでの時間を調べる
MS_ONTM	マウスボタンを押すまでの時間を調べる
MS_PATST	マウスカーソルのパターンを定義する
MS_SEL	マウスカーソルのパターンを選択する
MS_SEL2	マウスカーソルのパターンを複数指定してアニメーションを行う
SKEY_MOD	ソフトキーボードの制御を行う

3.17 IOCSコール

関数名	機 能
DENSNS	電卓のセンスを行う
ONTIME	IOCS が起動してから現在までの時間を得る
B_INTVCS	ベクタの設定を行う
B_SUPER	スーパーバイザモードへの切り替えを行う
B_BPEEK	指定アドレスからデータを読み込む (バイト単位)
B_WPEEK	指定アドレスからデータを読み込む (ワード単位)
B_LPEEK	指定アドレスからデータを読み込む (ロングワード単位)
B_MEMSTR	指定アドレスからデータを読み込む (バイト数指定)
B_BPOKE	指定アドレスへデータを書き込む (バイト単位)
B_WPOKE	指定アドレスへデータを書き込む (ワード単位)
B_LPOKE	指定アドレスへデータを書き込む (ロングワード単位)
B_MEMSET	指定アドレスへデータを書き込む (バイト数指定)
DMAMOVE	DMA 転送を行う
DMAMOV_A	DMA 転送を行う (アレイチェーンによる)
DMAMOV_L	DMA 転送を行う (リンクアレイチェーンによる)
DMAMODE	DMA の実行モードをセンスする
BOOTINF	ブート情報を得る
ROMVER	ROM のバージョンと作成年月日を得る

関数名	機 能
G_CLR_ON	グラフィック画面をクリアして、表示モードにする
GPALET	グラフィックのパレットを設定する
SFTJIS	シフト JIS コードを JIS 漢字コードに変換する
JISSFT	JIS 漢字コードをシフト JIS コードに変換する
AKCONV	ANK コードをシフト JIS コードに変換する
RMACNV	ローマ字変換を行う
DAKJOB	全角文字の濁点処理を行う
HANJOB	全角文字の半濁点処理を行う
OS_CURON	カーソルを表示する
OS_CUROF	カーソルを消す
APAGE	グラフィック画面の書き込みページを設定する
VPAGE	グラフィック画面の表示ページを設定する
HOME	グラフィック画面の表示位置を設定する
WINDOW	グラフィック画面にウィンドウを設定する
WIPE	グラフィック画面をクリアする
PSET	グラフィック画面に点を表示する
POINT	グラフィック画面上の点のパレットコードを得る
LINE	グラフィック画面にラインを引く
BOX	グラフィック画面にボックスを描く
FILL	グラフィック画面にボックスを描きその中を塗りつぶす
CIRCLE	グラフィック画面にサークルを描く
PAINT	グラフィック画面をペイントする
SYMBOL	グラフィック画面に文字を書く

3.17 IOCSコール

関数名	機 能
GETGRM	グラフィック画面の読み込みを行う
PUTGRM	グラフィック画面に書き込みを行う
SP_INIT	スプライト面の初期化を行う
SP_ON	スプライト面の表示を ON にする
SP_OFF	スプライト面の表示を OFF にする
SP_CGCLR	PCG をクリアする
SP_DEFCG	PCG を設定する
SP_GTPCG	PCG の読み出しを行う
SP_REGST	スプライトレジスタの設定を行う
SP_REGGT	スプライトレジスタの読み出しを行う
BGSCRLST	BG スクロールレジスタの設定を行う
BGSCRLGT	BG スクロールレジスタの読み出しを行う
BGCTRLST	BG コントロールレジスタの設定を行う
BGCTRLGT	BG コントロールレジスタの読み出しを行う
BGTEXTCL	BG テキストを指定のコードで埋める
BGTEXTST	BG テキストを設定する
BGTEXTGT	BG テキストの読み出しを行う
SPALET	スプライトパレットの設定と読み出しを行う
TXXLINE	テキスト画面に X 方向ラインを引く
TXYLINE	テキスト画面に Y 方向ラインを引く
TXBOX	テキスト画面にボックスを描く
TXFILL	テキスト画面にボックスを描きその中を塗りつぶす
TXREV	テキスト画面の反転を行う

3.17 IOCSコール

関数名	機 能
TXRASCPY	テキスト画面でラスタコピーを行う
ABORTRST	アボートするためのフラグをリセットする (OSで使用)
IPLERR	IPL 時のエラー処理を行う
ABORTJOB	アボート処理を行う (OSで使用)

関数名	機 能
EXIT	プログラムを終了する
GETCHAR	キーボードから1文字を読み取る
PUTCHAR	文字をキーボードに表示する
COMINP	VUX (RS-232C) から1文字を読み取る
COMOUT	VUX (RS-232C) へ1文字を送信する
PRINOUT	プリンタへ1文字を送信する
INPUT	キーボードから1文字を読み取る (アスキーコードあり)
INKEY	キーボードから1文字を読み取る (アスキーコードあり)
GETC	文字コードを読み取る (アスキーコードあり)
PRINT	文字列を表示する
GETS	文字列を読み取る (アスキーコードあり)

3.18 DOSコール

DOS コールは、Human68k のサブルーチンを呼び出すシステムコールのことであり、おもな機能は以下の通りです。

- RS-232C 回線の入出力制御
- キーボード入力制御
- 漢字変換行の制御
- コンソール直接入出力
- 日付、時刻の制御
- ファイルの入出力制御
- デバイスドライバの直接入出力
- プログラムのロードと実行
- 環境変数の制御

DOS コールの関数には以下のものがあり、これらはすべてインクルードファイル `doslib.h` で宣言されています。

関数名	機 能
EXIT	プログラムを終了する
GETCHAR	キーボードより文字コードを入力し、表示する
PUTCHAR	文字コードを表示する
COMINP	AUX (RS-232C) より 1 文字入力する
COMOUT	AUX (RS-232C) へ 1 文字出力する
PRNOUT	プリンタへ 1 文字出力する
INPOUT	キー入力/センスまたは文字コードの表示を行う
INKEY	文字コードを入力する (ブレイクチェックなし)
GETC	文字コードを入力する (ブレイクチェックあり)
PRINT	文字列を表示する
GETS	文字列を入力する (ブレイクチェックあり)

関数名	機 能
KEYSNS	キーの入力状態を調べる
KFLUSHGP	先行入力を捨ててキーボード入力を行う
KFLUSHIO	先行入力を捨ててキーボード入力を行う
KFLUSHIN	先行入力を捨ててキーボード入力を行う
KFLUSHGC	先行入力を捨ててキーボード入力を行う
KFLUSHGS	先行入力を捨ててキーボード入力を行う
FFLUSH	ディスクをリセットする
CHGDRV	カレントドライブを指定する
DRVCTRL	指定ドライブの状態のチェックと設定を行う
CONSNS	画面への出力が可能であるか否か調べる
PRNSNS	プリンタへの出力が可能であるか否か調べる
CINSNS	AUX (RS-232C) からの入力が可能であるか否か調べる
COUTSNS	AUX (RS-232C) への出力が可能であるか否か調べる
FATCHK	指定ファイルのドライブ番号とセクタの関連を調べる
FATCHK2	大容量の指定ファイルのドライブ番号とセクタの関連を調べる
HENDSPMO	漢字変換行の制御 (モード表示ウィンドウのオープン)
HENDSPMP	漢字変換行の制御 (モード表示のウィンドウへのノーマル表示)
HENDSPMR	漢字変換行の制御 (モード表示ウィンドウへのリバース表示)
HENDSPMC	漢字変換行の制御 (モード表示ウィンドウのクローズ)

3.18 DOSコール

関数名	機 能
HENDSPIO	漢字変換行の制御 (入力ウィンドウのオープン)
HENDSPIP	漢字変換行の制御 (入力ウィンドウのノーマル表示)
HENDSPIR	漢字変換行の制御 (入力ウィンドウのリバース表示)
HENDSPIC	漢字変換行の制御 (入力ウィンドウのクローズ)
HENDSPSO	漢字変換行の制御 (候補ウィンドウのオープン)
HENDSPSP	漢字変換行の制御 (候補ウィンドウへのノーマル表示)
HENDSPSR	漢字変換行の制御 (候補ウィンドウへのリバース表示)
HENDSPSC	漢字変換行の制御 (候補ウィンドウのクローズ)
CURDRV	カレントドライブを得る
GETSS	文字列を入力する (ブレークチェックなし)
FGETC	指定したファイルハンドルから1文字入力を行う
FGETS	指定したファイルハンドルから文字列を入力する
FPUTC	指定した文字コードをファイルハンドルへ出力する
FPUTS	指定した文字列をファイルハンドルへ出力する
ALLCLOSE	すべてのファイルハンドルをクローズする
SUPER	スーパーバイザモード/ユーザーモードの切り替えを行う
FNCKEYGT	再定義可能キーの読み出しを行う
FNCKEYST	再定義可能キーの設定を行う
C_PUTC	コンソール直接出力 (指定した1バイトデータの表示)

関数名	機 能
C_PRINT	コンソール直接出力 (指定した文字列データの表示)
C_COLOR	コンソール直接出力 (指定した属性の設定)
C_LOCATE	コンソール直接出力 (カーソル位置の設定)
C_DOWN_S	コンソール直接出力 (1行下へカーソル移動) (最下行のときスクロール UP)
C_UP_S	コンソール直接出力 (1行上へカーソル移動) (最上行のときスクロール DOWN)
C_UP	コンソール直接出力 (n行上へカーソル移動)
C_DOWN	コンソール直接出力 (n行下へカーソル移動)
C_RIGHT	コンソール直接出力 (n桁右へカーソル移動)
C_LEFT	コンソール直接出力 (n桁左へカーソル移動)
C_CLS_ED	コンソール直接出力 (カーソル位置から最終行右端までクリア)
C_CLS_ST	コンソール直接出力 (先頭行左端からカーソル位置までクリア)
C_CLS_AL	コンソール直接出力 (画面全体をクリア)
C_ERA_ED	コンソール直接出力 (カーソル位置から行の右端までクリア)
C_ERA_ST	コンソール直接出力 (行の左端からカーソル位置までクリア)
C_ERA_AL	コンソール直接出力 (カーソルのある行全体をクリア)
C_INS	コンソール直接出力 (カーソルのある行に n 行挿入)
C_DEL	コンソール直接出力 (カーソルのある行から n 行削除)

3.18 DOSコール

関数名	機 能
C_FNKMOD	コンソール直接出力 (ファンクションキー行のモード設定)
C_WINDOW	コンソール直接出力 (スクロール範囲の設定)
C_WIDTH	コンソール直接出力 (画面サイズの設定)
C_CURON	コンソール直接出力 (カーソル表示モードの設定)
C_CUROFF	コンソール直接出力 (カーソル非表示モードの設定)
K_KEYINP	コンソール直接入力 (キーボード入力)
K_KEYSNS	コンソール直接入力 (キーコードの先読み)
K_SFTSNS	コンソール直接入力 (シフトキー状態のセンス)
K_KEYBIT	コンソール直接入力 (キーコードグループの状態通知)
K_INSMOD	コンソール直接入力 (挿入モードの ON/OFF 設定)
INTVCS	ベクタにアドレスを設定する
PSPSET	プロセス管理情報を書き込む
GETTIM2	現在の時刻を求める
SETTIM2	時刻を設定する
NAMESTS	ファイル名に関する情報を得る
GETDATE	現在の日付を求める
SETDATE	日付を設定する
GETTIME	現在の時刻を求める
SETTIME	時刻を設定する

関数名	機 能
VERIFY	ベリファイフラグの設定を行う
DUP0	ファイルハンドルの強制複写
VERNUM	Human68k のバージョン番号を得る
KEEPPR	プログラムを常駐終了する
GETDPB	指定装置番号のドライブパラメータブロックをコピーする
BREAKCK	ブレイクチェックの設定を行う
DRVXCHG	ドライブ名の入れ替えを行う
INTVCG	ベクタの値を求める
DSKFRE	ディスクの残り容量を求める
NAMECK	ファイル名をバッファに展開する
MKDIR	ディレクトリを作成する
RMDIR	ディレクトリを削除する
CHDIR	カレントディレクトリを変更する
CREATE	ファイルを作成する
OPEN	ファイルをオープンする
CLOSE	ファイルをクローズする
READ	ファイルから読み込みを行う
WRITE	ファイルへ書き出しを行う
DELETE	ファイルを削除する
SEEK	ファイルポインタを移動する
CHMOD	ファイルの属性を変更する
IOCTRLGT	デバイスドライバの直接入出力（装置情報の通知）

3.18 DOSコール

関数名	機能
IOCTRLST	デバイスドライバの直接入出力（装置情報の設定）
IOCTRLRH	デバイスドライバの直接入出力（データの読み込み）
IOCTRLWH	デバイスドライバの直接入出力（データの書き出し）
IOCTRLRD	デバイスドライバの直接入出力（装置番号からの読み込み）
IOCTRLWD	デバイスドライバの直接入出力（装置番号からの書き出し）
IOCTRLIS	デバイスドライバの直接入出力（入力ステータスの通知）
IOCTRLOS	デバイスドライバの直接入出力（出力ステータスの通知）
IOCTRLDVGT	指定ドライブがローカルかリモートかを調べる
IOCTRLFDGT	指定ファイルがローカルかリモートかを調べる
IOCTRLRTSET	リトライ数を設定する
IOCTRLDVCTL	指定ドライブの特殊コントロールを行う
IOCTRLFDCTL	指定ファイルの特殊コントロールを行う
DUP	ファイルハンドルを複製する
DUP2	ファイルハンドルの強制複製を行う
CURDIR	カレントディレクトリを得る
MALLOC	メモリ領域を確保する
MFREE	メモリ領域を解放する
SETBLOCK	メモリブロックのサイズを変更する

関数名	機 能
EXEC2	プログラムのロード、実行、モジュール番号の取得を行う
LOADEXEC	プログラムをロードし実行する
LOAD	プログラムをロードする
PATHCHK	プログラムロードのための準備作業を行う
LOADONLY	プログラムをロードする
EXECONLY	ロード後のプログラムを実行する
BINDNO	プログラムのモジュール番号の取得を行う
EXIT2	プログラムを終了する
WAIT	プロセスの終了コードを得る
FILES	属性に従ってファイルを検索する
NFILES	属性に従って次のファイルを検索する
SETPDB	指定したプロセスに制御をわたす
GETPDB	現在のプロセスのアドレスを求める
SETENV	指定された環境に変数を設定する
GETENV	指定された環境から指定された変数の内容を取り出す
VERIFYG	ベリファイフラグの設定状況調べる
COMMON_CK	コモン領域が存在するかどうか調べる
COMMON_RD	コモン領域からデータの読み込みを行う
COMMON_WT	コモン領域の作成およびデータの書き込みを行う
COMMON_LK	コモン領域をロックする
COMMON_FRE	コモン領域のロックを削除する
COMMON_DEL	コモン領域を削除する
MOVE	ファイルを移動する

3.18 DOSコール

関数名	機 能
RENAME	ファイル名を変更する
FILEDATE	ファイルの作成日付／時刻の読み出しと設定
MALLOC2	割り当てのモードを指定してメモリを割り当てる
MAKETMP	作業用ファイルを作成する
NEWFILE	新規にファイルを作成する
LOCK	ファイルの共有を禁止する
UNLOCK	ファイルの共有の禁止を解除する
GETASSIGN	仮想割り当ての状態を得る
MAKEASSIGN	仮想割り当てを作成する
RASSIGN	仮想割り当てを解除する
S_MALLOC	主スレッドのメモリからメモリを割り当てる
S_MFREE	主スレッドのメモリから割り当てたメモリを開放する
S_PROCESS	サブスレッドのメモリ領域を設定する
DISKRED	ブロックデバイスの直接リード
DISKRED2	大容量ブロックデバイスの直接リード
DISKWRT	ブロックデバイスの直接ライト
DISKWRT2	大容量ブロックデバイスの直接ライト
INDOSFLG	OS の内部ワークのアドレスを得る
SUPER_JSR	スーパーバイザ領域のサブルーチンを呼び出す
BUS_ERR	バスエラーが発生するかを調べ、メモリ転送を行う
OPEN_PR	サブスレッドを作成する
KILL_PR	サブスレッドを削除する
GET_PR	スレッド情報を得る

関数名	機 能
SUSPEND_PR	スレッドを強制的にスリープ状態にする
SLEEP_PR	スレッドをスリープ状態にする
SEND_PR	スレッド間のコマンド、データ転送を行う
TIME_PR	スレッド管理用のタイマーの値を得る
CHANGE_PR	スレッドの実行権を放棄する

3.19 その他

これらの関数は、他のカテゴリーには分類できないものです。機能のカッコ内は、宣言されているインクルードファイル名です。

関数名	機 能
abs	int 型整数値の絶対値を求める (stdlib. h)
labs	long 型整数値の絶対値を求める (stdlib. h)
wabs	short 型整数値の絶対値を求める (stdlib. h)
assert	論理エラーをテストする (assert. h)
getenv	環境変数の値を得る (stdlib. h)
longjmp	セーブされたスタック環境をリストアする (setjmp. h)
max	2 つの数の大きい方を求める (stdlib. h)
mix	2 つの数の小さい方を求める (stdlib. h)
offsetof	構造体のメンバの位置を得る (stddef. h)
perror	エラーメッセージを表示する (stdio. h)
strerror	エラーメッセージを得る (string. h)
putenv	環境変数の値を修正／追加する (stdlib. h)
srand	疑似乱数を初期化する (stdlib. h)
rand	疑似乱数を得る (stdlib. h)
setjmp	スタック環境をセーブする (setjmp. h)
va_arg	可変引数の内容を取り出す (stdarg. h)
va_end	可変引数の取り出しを終わる (stdarg. h)
va_start	可変引数の取り出しを始める (stdarg. h)

リファレンス

章 4 第

この章では、XC コンパイラがサポートするライブラリ関数（小文字）を、リファレンス形式でアルファベット順に掲載しています。

関数には、レベルが設定されています。

これは、C 言語で一般的にいわれている入出力のレベルとは違い、XC コンパイラ独自のものです。

レベルが表す意味は、次の通りです。

レベル 3 : BASIC 関連の関数（ライブラリファイル BASLIB. L に含まれる）のうち、他のレベルの関数と混在させて使うときには、注意しなければならないもの。

レベル 2 : ライブラリファイル CLIB. L に含まれるすべての関数、および、これらの関数と混在させて使用できるもの。

レベル 1 : DOS コール関連の関数（ライブラリファイル DOSLIB. L に含まれる）のうち、他のレベルの関数と混在させて使うときには、注意しなければならないもの。

レベル 0 : IOCS コール関連の関数（ライブラリファイル IOCSLIB. L に含まれる）のうち、他のレベルの関数と混在させて使うときには、注意しなければならないもの。

レベル 2 以外の関数を使用するときは、なるべく同一レベルの関数だけでプログラムを記述してください。

abort

レベル 2

書 式 #include <stdlib.h>

```
void abort( );
```

機 能

現在実行中のプロセスを終了し、呼び出し元のプロセス（通常オペレーティングシステム）に戻ります。

このとき、呼び出し元のプロセスには終了ステータス 3 が返されます。

また、次のメッセージを標準エラー出力 (stderr) に出力します。

Abnormal program termination

なお、abort を呼び出してもストリームバッファの内容はフラッシュされません。

戻 り 値 戻り値はありません。

参 照 関 数 exit, _exit

プログラム例

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

main(int argc, char *argv[])
{
    char *name;
    FILE *fp;

    name = (argc > 1) ? argv[1] : "file";
    if ((fp = fopen(name, "r")) == NULL) {
        perror(name);
        abort();
    }
    printf("open succeeded\n");
    (void)fclose(fp);
}
```

abs, labs, wabs

レベル 2

書 式 #include <stdlib.h>

```
int abs(n) ; /* int 型の整数値の絶対値を求める */
int n; /* int 型の整数値 */

long labs(n) ; /* long 型の整数値の絶対値を求める */
long n; /* long 型の整数値 */

short wabs(n) ; /* short 型の整数値の絶対値を求める */
short n; /* short 型の整数値 */
```

機 能 整数 n の絶対値を求めます。

戻 り 値 整数 n の絶対値を返します。

参 照 関 数 cabs, fabs

プログラム例

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

```
main()
{
```

```
    int    i;
    long    l;
```

```
    for (i = -10; i <= 10; i++)
        printf("abs(%d) = %d\n", i, abs(i));
```

```
    for (l = -10L; l <= 10L; l++)
        printf("labs(%ld) = %ld\n", l, labs(l));
```

```
}
```

access

レベル 2

書式

#include <io.h>

```
int access(pathname, mode) ;
char *pathname; /* ファイルのパス名 */
int mode; /* ファイルのアクセスモード */
```

機能

pathname で指定されたファイルが mode で指定したモードでアクセスできるかどうかを調べます。

mode の値とその内容を以下に示します。

mode の値	内 容
06	読み込み、書き込みともに可能かどうかを調べる
04	読み込み可能かどうかを調べる
02	書き込み可能かどうかを調べる
00	ファイルが存在するかどうかを調べる

Human68k では、ファイルとして存在すれば必ず読むことができますので、00 と 04 は同じ結果になります。

同様に 02 と 06 も同じ結果になります。

戻り値

ファイルが指定したアクセスモードをもつときには 0 を返します。

ファイルが存在しないとき、または指定したモードではアクセスできないときには -1 が返され、同時に errno に次の値のうちのいずれかがセットされます。

errno の値	内 容
EACCES	mode で指定したアクセスはできない
ENOENT	ファイル名またはパス名が見つからない

参照関数

chmod, fstat, open, stat

プログラム例

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

main()
{
    char *filename = "file";

    if ((access(filename, 0)) != 0)
        printf("%s: file が存在しない。%n", filename);
    else {
        if ((access(filename, 4)) == 0)
            printf("%s: 読み込み可能%n", filename);
        else
            printf("%s: 読み込み不可%n", filename);

        if ((access(filename, 2)) == 0)
            printf("%s: 書き込み可能%n", filename);
        else
            printf("%s: 書き込み不可%n", filename);
    }
}
```

HumanOSでは、ファイルとして存在しないファイルに対しては、00
と04は同じ結果になります。
同様に02と06も同じ結果になります。

ファイルが存在しない状態でアクセスを試みると、エラーコードが返ります。
また、ファイルが存在しない状態でアクセスを試みると、エラーコードが返ります。
これは、同時にerrnoに次の値のうちのいずれかが返ります。

errno の値	内容
ENOTDIR	ファイル名またはパス名がディレクトリでない
EINVAL	modeで指定したアクセスモードが不正

chmod, fstat, open, stat

a_cont

レベル 3

書 式 #include <audio.h>

void a_cont();

機 能 中断されていた PCM の動作を再開します。

戻 り 値 戻り値はありません。

参 照 関 数 a_end, a_play, a_rec, a_stat, a_stop

プログラム例

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

#define KHz3_9      0
#define KHz5_2      1
#define KHz7_8      2
#define KHz10_4     3
#define KHz15_6     4
#define M_CUT       0
#define M_LEFT      1
#define M_RIGHT     2
#define M_STEREO    3
#define SIZE        32732
#define A_SIZE      65536
#define NASI        'NASI'
#define ST_NON      0 /* 何も実行していない */
#define ST_OUT      2 /* 出力中(ADPCMOUT, a_play) */
#define ST_INP      4 /* 入力中(ADPCMINP, a_rec) */
#define ST_AOT      12 /* 出力中(ADPCMAOT) */
#define ST_AIN      14 /* 入力中(ADPCMAIN) */
#define ST_LOT      22 /* 出力中(ADPCMLOT) */
#define ST_LIN      24 /* 入力中(ADPCMLIN) */

static char aca[A_SIZE];

main()
{
    a_rec(aca, A_SIZE, KHz15_6, NASI);
    while (a_stat() != ST_NON)
        puts("recording...");

    a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
    if (a_stat() != ST_NON) {
        puts("playing...");
        a_stop();
        puts("paused...");
        a_cont();
        puts("contune...");
    }
}
```

acos

レベル 2

書 式 #include <math.h>

```
double acos(x) ;
double x ;
```

機 能 x ($-1 \leq x \leq 1$) のアークコサインを計算します。

戻 り 値 x のアークコサインの値 ($0 \sim \pi$) を返します。
 x が -1 より小さいか 1 より大きい場合は 0 を返し、errno に EDOM がセットされます。
 エラー処理は matherr ルーチンで行うことができます。

参 照 関 数 asin, atan, atan2, cos, matherr, sin, tan

プログラム例

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

main()
{
    double x;
    int i;

    for (i = -10; i <= 10; i++) {
        x = (double)i / 10.0;
        printf("acos(%+4.1f) = %g\n", x, acos(x));
    }
}
```

a_end

レベル 3

書 式 `#include <audio.h>``void a_end() ;`

機 能 PCM の動作を強制終了します。

戻 り 値 戻り値はありません。

参 照 関 数 `a_cono, a_play, a_rec, a_stat, a_stop`

プログラム例

```

#include <stdio.h>
#include <audio.h>

#define KHz3_9      0
#define KHz5_2      1
#define KHz7_8      2
#define KHz10_4     3
#define KHz15_6     4
#define M_CUT       0
#define M_LEFT      1
#define M_RIGHT     2
#define M_STEREO    3
#define SIZE        32732
#define A_SIZE      65536
#define NASI        'NASI'

#define ST_NON      0      /* 何も実行していない */
#define ST_OUT      2      /* 出力中(ADPCMOUT, a_play) */
#define ST_INP      4      /* 入力中(ADPCMINP, a_rec) */
#define ST_AOT      12     /* 出力中(ADPCMAOT) */
#define ST_AIN      14     /* 入力中(ADPCMMAIN) */
#define ST_LOT      22     /* 出力中(ADPCMLOT) */
#define ST_LIN      24     /* 入力中(ADPCMMLIN) */

static char aca[A_SIZE];

main()
{
    a_rec(aca, A_SIZE, KHz15_6, NASI);
    while (a_stat() != ST_NON)
        puts("recording...");

    a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
    if (a_stat() != ST_NON) {
        puts("playing...");
        a_end();
        puts("aborted...");
    }
}

```

apage

レベル 3

書式 #include <graph.h>

```
void apage(page) ;
char page; /* アクティブページ */
```

機能 グラフィック画面のアクティブページを設定します。

なお、ページ数は画面のモードによって異なります。

戻り値 戻り値はありません。

参照関数 screen, vpage

プログラム例

```
#include <basic0.h>
#include <graph.h>

main()
{
    int i;

    b_init();

    screen(1, 1, 1, 1);
    vpage(0);

    apage(0);
    fill(50, 50, 200, 200, 3);
    apage(1);
    fill(100, 100, 250, 250, 5);
    apage(2);
    fill(150, 150, 300, 300, 9);
    apage(3);
    fill(200, 200, 350, 350, 15);

    for (i = 0; i < 10000; i++)
        vpage(0x01);
    for (i = 0; i < 10000; i++)
        vpage(0x02);
    for (i = 0; i < 10000; i++)
        vpage(0x04);
    for (i = 0; i < 10000; i++)
        vpage(0x08);
    vpage(0x0f);

    b_exit(0);
}
```

a_play

レベル 3

書 式 #include <audio.h>

関数プロトタイプ

```
void a_play(buf, size, cl, md, length) ;
void *buf; /* PCM データ格納配列へのポインタ */
int size; /* buf のバイトサイズ */
int cl; /* サンプリング周波数 */
int md; /* 出力モード */
int length; /* 有効バイト数 */
```

機 能 PCM 録音の再生を行います。

buf には PCM データ格納配列のアドレスを指定します。

size には buf のバイト数を指定します。

cl にはサンプリング周波数を指定します。

0 : 3.9KHz

1 : 5.2KHz

2 : 7.8KHz

3 : 10.4KHz

4 : 15.6KHz

md には出力モードを指定します。

0 : カット

1 : 左

2 : 右

3 : ステレオ (左右両方)

length には有効バイト数を指定します。指定したところまでが再生されます。

戻 り 値 戻り値はありません。

プログラム例

```
#include <stdio.h>
#include <conio.h>
#include <audio.h>

#define KHz3_9 0
#define KHz5_2 1
#define KHz7_8 2
#define KHz10_4 3
#define KHz15_6 4
#define M_CUT 0
#define M_LEFT 1
#define M_RIGHT 2
#define M_STEREO 3
#define SIZE 32732
#define A_SIZE 65536
#define NASI 'NASI'

static char aca[A_SIZE];

main()
{
    hit_any_key("PCM record?");
    a_rec(aca, A_SIZE, KHz15_6, NASI);

    hit_any_key("PCM play?");
    a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
}

hit_any_key(char *str)
{
    printf("%s hit any key\n", str);
    (void)getch();
}
```

asc

レベル 3

書 式 #include <basic.h>

```
int asc(s) ;
```

```
char *s; /* 文字列 */
```

機 能 文字列 s の先頭 1 バイトの文字コードを返します。

戻 り 値 文字列 s の先頭 1 バイトの文字コードを返します。

a_rec

レベル 3

書 式 #include <audio. h>

```
void a_rec(buf, size, cl, length) ;  
void *buf; /* PCM データ格納配列へのポインタ */  
int size; /* buf のバイトサイズ */  
int cl; /* サンプリング周波数 */  
int length; /* 有効バイト数 */
```

機 能

PCM 録音を行い、そのデータを配列に格納します。

buf には PCM データを格納する配列のアドレスを指定します。

size には buf のバイト数を指定します。

cl にはサンプリング周波数を指定します。

0 : 3.9KHz

1 : 5.2KHz

2 : 7.8KHz

3 : 10.4KHz

4 : 15.6KHz

length には有効バイト数を指定します。

指定したところまで録音します。

戻 り 値

戻り値はありません。

プログラム例

```

#include      <stdio.h>
#include      <conio.h>
#include      <audio.h>

#define KHz3_9      0
#define KHz5_2      1
#define KHz7_8      2
#define KHz10_4     3
#define KHz15_6     4
#define M_CUT        0
#define M_LEFT       1
#define M_RIGHT      2
#define M_STEREO     3
#define SIZE         32732
#define A_SIZE       65536
#define NASI         'NASI'

static char      aca[A_SIZE];

main()
{
    hit_any_key("PCM record?");
    a_rec(aca, A_SIZE, KHz15_6, NASI);

    hit_any_key("PCM play?");
    a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
}

hit_any_key(char *str)
{
    printf("%s hit any key\n", str);
    (void)getch();
}

```

asctime

レベル 2

書 式 `#include <time.h>`

```
char *asctime(time) ;  
struct tm *time; /* tm 構造体を指すポインタ */
```

機 能 tm 構造体に格納されている時間情報を、文字列に変換します。
得られる文字列は 26 文字で、フォーマットは以下のようになります。

wed Oct 22 13:15:30 1986¥n¥0

なお、時刻は 24 時間表示です。

asctime は、内部の静的な領域へのポインタを返しますので、返されたポインタが指す領域を書き変えてはいけません。

また、前回の呼び出しで返された文字列は失われます。

戻 り 値 変換された文字列へのポインタを返します。

参 照 関 数 ctime, ftime, gmtime, time, tzset

プログラム例

```
#include <stdio.h>  
#include <time.h>  
  
main()  
{  
    struct tm      *ntime;  
    time_t  ltime;  
  
    time(&ltime);  
    ntime = localtime(&ltime);  
    printf("ただ今の時刻は %s", asctime(ntime));  
}
```

asin

レベル 2

書 式 `#include <math.h>`

`double asin(x);`

`double x;`

機 能 `x` ($-1 \leq x \leq 1$) のアークサインを計算します。

戻 り 値 `x` のアークサインの値 ($-\pi/2 \sim \pi/2$) を返します。エラー発生時に `x` が -1 より小さいか 1 より大きい場合は `0` を返し、`errno` に `EDOM` がセットされます。エラー処理は `matherr` 関数を使って行うことができます。

参 照 関 数 `acos`, `atan`, `atan2`, `cos`, `matherr`, `sin`, `tan`

プログラム例

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

main()
{
    double x;
    int i;

    for (i = -10; i <= 10; i++) {
        x = (double)i / 10.0;
        printf("asin(%.1f) = %g\n", x, asin(x));
    }
}
```

assert

レベル 2

書 式 `#include <assert.h>`
 `#include <stdio.h>`

`void assert(expression) ; /* expression=論理式 */`

機 能 `assert` マクロは、`expression` が偽 (0) の場合に、診断メッセージを出力し、プロセスを終了します。

診断メッセージは下記の形式です。

Assertion failed: expression, file filename, line linenumber

`expression` は論理式、`filename` はソースファイル名で、`linenumber` は `expression` が偽になった行番号です。

`expression` が真 (0 以外) のときは、何も行いません。

通常 `assert` は、デバッグ時にプログラムの論理エラーの判定のために使用します。

デバッグの終了後、`NDEBUG` を使って `assert` をプログラムから取り除きます。`NDEBUG` をコマンド行のオプション/Dで指定するか、`#define` プリプロセッサ命令を使って定義すると、コンパイル時にプリプロセッサによりすべての `assert` 呼び出しがソースプログラムから取り除かれます。

戻 り 値 戻り値はありません。

プログラム例

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

main()
{
    check_string("str", 3);
    check_string(NULL, 0);
    check_string("", 3);
    check_string("str", 0);
}

check_string(char *str, int len)
{
    assert(str != NULL);
    assert(*str != '\0');
    assert(len > 0);
}
```

a_stat

レベル 3

書 式 #include <audio.h>

int a_stat();

機 能 現在の PCM の動作状況を調べ、戻り値として返します。

戻 り 値 PCM の実行状態を以下に示す値で返します。

0x00 : 何も実行していない
 0x02 : 出力中 (ADPCMOUT, a_play を実行中)
 0x04 : 入力中 (ADPCMINP, a_rec を実行中)
 0x12 : 出力中 (ADPCMAOT を実行中)
 0x14 : 入力中 (ADPCMAIN を実行中)
 0x22 : 出力中 (ADPCMLOT を実行中)
 0x24 : 入力中 (ADPCMLIN を実行中)

なお、a_stat の戻り値は、ADPCMSNS 関数の戻り値と同じです。

参 照 関 数 a_cont, a_end, a_play, a_rec, a_stop, ADPCMSNS

プログラム例

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

#define KHz3_9 0
#define KHz5_2 1
#define KHz7_8 2
#define KHz10_4 3
#define KHz15_6 4
#define M_CUT 0
#define M_LEFT 1
#define M_RIGHT 2
#define M_STEREO 3
#define SIZE 32732
#define A_SIZE 65536
#define NASI 'NASI'
#define ST_NON 0 /* 何も実行していない */
#define ST_OUT 2 /* 出力中 (ADPCMOUT, a_play) */
#define ST_INP 4 /* 入力中 (ADPCMINP, a_rec) */
#define ST_AOT 12 /* 出力中 (ADPCMAOT) */
#define ST_AIN 14 /* 入力中 (ADPCMAIN) */
#define ST_LOT 22 /* 出力中 (ADPCMLOT) */
#define ST_LIN 24 /* 入力中 (ADPCMLIN) */

static char aca[A_SIZE];

main()
{
    a_rec(aca, A_SIZE, KHz15_6, NASI);
    while (a_stat() != ST_NON)
        puts("recording...");

    a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
    while (a_stat() != ST_NON)
        puts("playing...");
}
```

a_stop

レベル 3

書 式 #include <audio.h>

void a_stop();

機 能 PCM の動作を中断します。

a_cont 関数で再開することができます。

戻 り 値 戻り値はありません。

参 照 関 数 a_cont, a_end, a_play, a_rec, a_stat

プログラム例

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

#define KHz3_9      0
#define KHz5_2      1
#define KHz7_8      2
#define KHz10_4     3
#define KHz15_6     4
#define M_CUT       0
#define M_LEFT      1
#define M_RIGHT     2
#define M_STEREO    3
#define SIZE        32732
#define A_SIZE      65536
#define NASI        'NASI'

#define ST_NON      0 /* 何も実行していない */
#define ST_OUT      2 /* 出力中 (ADPCMOUT, a_play) */
#define ST_INP      4 /* 入力中 (ADPCMINP, a_rec) */
#define ST_AOT      12 /* 出力中 (ADPCMAOT) */
#define ST_AIN      14 /* 入力中 (ADPCMAIN) */
#define ST_LOT      22 /* 出力中 (ADPCMLOT) */
#define ST_LIN      24 /* 入力中 (ADPCMLIN) */

static char aca[A_SIZE];

main()
{
    a_rec(aca, A_SIZE, KHz15_6, NASI);
    while (a_stat() != ST_NON)
        puts("recording...");

    a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
    if (a_stat() != ST_NON) {
        puts("playing...");
        a_stop();
        puts("paused...");
        a_cont();
        puts("contune...");
    }
}
```

atan, atan2

レベル 2

書 式 #include <math.h>

```
double atan(x) ;  
double x ;
```

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

機 能 atan は x のアーктanジェントを計算します。
atan2 は x/y のアーктanジェントを計算します。

戻 り 値 atan は x のアークトanジェントの値 ($-\pi/2 \sim \pi/2$) を、atan2 は x/y のアークトanジェントの値 ($-\pi \sim \pi$) をそれぞれ返します。
引数が許される範囲にない場合は、返り値は不定となり、errno に EDOM がセットされます。
エラー処理は matherr 関数を使って行うことができます。

参 照 関 数 acos, asin, cos, matherr, sin, tan

プログラム例

```
#include <stdio.h>  
#include <math.h>  
  
main()  
{  
    double x;  
    double y;  
    int i;  
    int j;  
  
    for (i = -32; i <= 32; i++) {  
        x = (double)i / 10.0;  
        printf("atan(%+4.1f) = %g\n", x, atan(x));  
    }  
    for (i = -32; i <= 32; i++) {  
        for (j = -32; j <= 32; j++) {  
            x = (double)i / 10.0;  
            y = (double)j / 10.0;  
            printf("atan2(%+4.1f, %+4.1f) = %g\n",  
                y, x, atan2(y, x));  
        }  
    }  
}
```


atexit

レベル 2

書 式 #include <stdlib.h>

```
int atexit(void(* func)());
void (* func)(); /* 終了時に呼び出される関数へのポインタ */
```

機 能 func で指定した関数を、プログラム終了時に呼び出す関数として登録します。
exit によってプログラムが正常終了すると、呼び出し元の親プロセスに戻る前に(* func)()を呼び出します。

呼び出される関数は、引数なしで、戻り値を持たない関数です。

atexit が呼び出されるたびに、新たな関数を追加・登録します。

複数の関数が登録された場合、終了時に実行される順序は登録された順番と逆の順番です。

登録できる関数の個数は最大 32 個までです。

戻 り 値 成功すると 0 を、失敗すると (32 個を超えた場合など) 0 以外を返します。

参 照 関 数 exit

プログラム例

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

void func1(void)
{
    printf("終了時の処理その1\n");
}

void func2(void)
{
    printf("終了時の処理その2\n");
}

main()
{
    if (atexit(func1) != 0)
        printf("登録できません。%\n");
    if (atexit(func2) != 0)
        printf("登録できません。%\n");
    exit(0);
}
```

atof, atoi, atol, atow

レベル 2

書 式 #include <stdlib.h>

```
double atof(string) ;
int atoi(string) ;
long atol(string) ;
short atow(string) ;
char *string; /* 変換される文字列へのポインタ */
```

機 能 atof は文字列 string を倍精度 (double 型) の浮動小数点値に変換します。

atoi は文字列 string を整数値に変換します。

atol は文字列 string を long 型整数値に変換します。

atow は文字列 string を short 型整数値に変換します。

string は数値に変換できる一連の文字列で、数値または数値の一部として解釈できない最初の文字までが変換の対象になります。

変換できる文字列の形式を以下に示します。

[atof]

[空白 (またはタブ文字)] [符号] [数字] [. 数字] [E [符号] 数字]

[空白 (またはタブ文字)] は無視されます。

[符号] は “+” または “-” です。“+” のときは省略できます。

[数字] は 1 桁以上の 10 進数です。これがないときは、[. 数字] に少なくとも 1 桁の数字が必要です。

[. 数字] は小数点以下の数字です。

また、-123E-12 のように指数表現することもできます。

[atoi, atol, atow]

[空白 (またはタブ文字)] [符号] 数字

各々の意味は atof と同様です。

戻 り 値

それぞれ変換した数値を返します。

変換できなかった場合は 0 を返します。

またオーバーフローした場合の戻り値は不定です。

プログラム例

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

main()
{
    printf("%g\n", atof("1.49597870e+11"));
    printf("%d\n", atoi("09471"));
    printf("%ld\n", atol("037971030"));
    printf("%d\n", atow("10287"));
}
```

b_binS

レベル 3

書 式 #include <basic. h>

```
char *b_binS(s, n) ;  
char *s; /* 文字列格納領域へのポインタ */  
int n; /* 数値 */
```

機 能 数値 n を 2 進数表現の文字列に変換し、s が指す領域へ格納します。
s が指す領域は、最低 34 バイト以上なければなりません。

戻 り 値 変換結果の文字列へのポインタを返します。

b_chrs

レベル 3

書 式 #include <basic. h>

```
char *b_chrs(s, ch) ;
char *s; /* 文字列格納領域へのポインタ */
int ch; /* 文字コード */
```

機 能 文字コードを文字列に変換します。

戻 り 値 変換結果の文字列へのポインタを返します。

```
#include <basic.h>

main()
{
    char a[10];
    b_init();
    b_inkey(a);
    b_print("caw off");
    b_print(STRCRLF);
    b_caw(0);
    b_inkey(a);
    b_print("caw on");
    b_print(STRCRLF);
    b_caw(1);
    b_inkey(a);
    b_exit(0);
}
```

b_csw

レベル 3

書 式 #include <basic0.h>

```
void b_csw(n) ;  
int n; /* on/off 指定 */
```

機 能 カーソルスイッチの on/off の指定を行います。

n には次の値を指定します。

0: off

1: on

戻 り 値 戻り値はありません。

プログラム例

```
#include <basic0.h>  
  
main()  
{  
    char a[4];  
  
    b_init();  
    b_inkeys(a);  
    b_sprint("csw off");  
    b_sprint(STRCRLF);  
    b_csw(0);  
    b_inkeys(a);  
    b_sprint("csw on");  
    b_sprint(STRCRLF);  
    b_csw(1);  
    b_inkeys(a);  
    b_exit(0);  
}
```

b_dateS

レベル 3

書 式 #include <basic0. h>

```
char *b_dateS(s) ;  
char *s; /* 日付文字列格納領域へのポインタ */
```

機 能 システムクロックの日付を“yy/mm/dd”の形式の文字列にして、sが指す領域に格納します。

戻 り 値 作成した文字列へのポインタを返します。

b_dayS

レベル 3

書 式 #include <basic0. h>

```
char *b_dayS(s) ;  
char *s; /* 曜日文字列格納領域へのポインタ */
```

機 能 システムクロックの曜日を漢字1文字（“日”、“月”、“火”、“水”、“木”、“金”、“土”）の文字列にして、sが指す領域に格納します。

戻 り 値 作成した文字列へのポインタを返します。

beep

レベル 3

書 式 `#include <basic0. h>`

`void beep( ) ;`

機 能 ビープ音を鳴らします。

戻 り 値 戻り値はありません。

b_exit

レベル 3

書 式 #include <basic0. h>

```
void b_exit(val) ;  
int val; /*終了コード*/
```

機 能

すべての環境をプロセスが起動したときの状態に戻して、親プロセスに戻ります。

オープンされているファイルはすべてクローズされます。

val の数値は、終了コードとして親プロセスにわたされます。

本関数を使用する場合は、この関数が呼び出される前に b_init 関数が呼び出されていなければいけません。

戻 り 値

戻り値はありません。

b_fclose

レベル 3

書 式 #include <basic. h>

```
int b_fclose(fp) ;  
int fp; /* 該当ファイルのファイル番号 */
```

機 能 fp で指定したファイルをクローズします。

戻 り 値 正常にクローズした場合は 0 を返します。
オープンしていないファイルをクローズしようとしたときは、-2 を返します。
その他のエラーのときには、-1 を返します。

b_fcloseall

レベル 3

書 式 #include <basic.h>

int b_fcloseall();

機 能 b_fopen 関数でオープンされたファイルをすべてクローズします。

戻 り 値 クローズしたファイル数を返します。

b_eof

レベル 3

書 式	#include <basic.h>
-----	--------------------

```
int b_eof(fp) ;  
int fp; /* 該当ファイルのファイル番号 */
```

機 能	fp で指定したファイルがエンドオブファイル (EOF) に達したか否かを調べます。
-----	--

戻 り 値	エンドオブファイルでない場合は 0 を返します。 オープンしていないファイルを指定したときは、-2 を返します。 エンドオブファイルの場合は、上記以外の値を返します。
-------	---

b_fgetc

レベル 3

書 式 #include <basic. h>

```
int b_fgetc(fp) ;  
int fp; /* 該当ファイルのファイル番号 */
```

機 能 fp で指定したファイルから 1 文字読み込みます。

戻 り 値 正常に読み込んだ場合、そのコードを返します。
オープンしていないファイルを指定したときは、-2 を返します。
その他のエラーのときには、-1 を返します。

b_flprint, b_ilprint, b_slprint, b_tlprint

レベル 3

書 式 #include <basic0. h>

```
void b_flprint(dval) ;
void b_ilprint(ival) ;
void b_slprint(str) ;
void b_tlprint(tabs) ;
```

```
double dval;
int ival;
char *str;
int tabs;
```

機 能 BASIC から C に変換したときに LPRINT 文は、LPRINT の引数によって次のそれぞれの関数に展開されます。

引数の型	関数
double	b_flprint (浮動小数点数)
int	b_ilprint (整数)
char *	b_slprint (文字列)

また、BASIC の TAB 関数を含んだ LPRINT 文は、b_tlprint に展開されます。
tabs はタブスキップの数です。
それぞれの引数をプリンタに出力します。

戻 り 値 戻り値はありません。

b_fopen

レベル 3

書 式 #include <basic.h>

```
int b_fopen(name, mode) ;  
char *name; /* ファイル名格納領域へのポインタ */  
char *mode; /* アクセスモード */
```

機 能 name が指すファイルをオープンします。
アクセスモードは、次の通りです。

mode	内 容
"r"	読み込み用にオープン
"w"	書き出し用にオープン
"rw"	読み書き用にオープン
"c"	新規にオープン

戻 り 値 ファイル番号を返します。
エラーのときは負の数を返します。

参 照 関 数 b_fwrites, b_fclose, b_fread

b_fprint

レベル 3

書 式 `#include <basic0. h>`

```
void b_fprint(f) ;
double f ;
```

機 能 浮動小数点数値 f をテキスト画面に表示します。

なお、整数表現が可能な数値には小数点がつきません。

戻 り 値 戻り値はありません。

b_fputc

レベル 3

書 式 #include <basic.h>

```
int b_fputc(ch, fp) ;  
char ch; /* 出力する文字 */  
int fp; /* 該当ファイルのファイル番号 */
```

機 能 文字 ch を、fp で指定したファイルに書き出します。

戻 り 値 書き出した文字コードを返します。
オープンしていないファイルを指定したときは、-2 を返します。
その他のエラーのときには、-1 を返します。

b_fread

レベル 3

書 式 #include <basic.h>

```
int b_fread(buff, b_size, count, fp) ;  
void *buff; /* 読み込み領域へのポインタ */  
int b_size; /* 読み込む項目のサイズ */  
int count; /* 読み込む項目数 */  
int fp; /* 該当ファイルのファイル番号 */
```

機 能 fp で指定したファイルから、b_size×count バイト数分読み込み、buff が指すバッファへ格納します。

戻 り 値 実際に読み込んだ項目数を返します。
オープンしていないファイルを読もうとしたときは、-2 を返します
その他のエラーのときには、-1 を返します。

参 照 関 数 b_fopen, b_fclose, b_fwite

b_freads

レベル 3

書 式 #include <basic.h>

```
int b_freads(buff, size, fp);  
char *buff; /* 読み込み領域へのポインタ */  
int size; /* 読み込み領域のバイト数 */  
int fp; /* 該当ファイルのファイル番号 */
```

機 能

ファイルから 1 行分のデータをバッファ buff に読み込みます。

バッファがいっぱいになるか、EOF、改行をみつけると、読み込みを終了します。

戻 り 値

実際に読み込んだ文字数を返します。

オープンしていないファイルを読もうとしたときは、-2 を返します。

その他のエラーのときには、-1 を返します。

参 照 関 数

b_fwrites, b_fopen, b_fclose

b_free

レベル 3

書 式 #include <basic0. h>

int b_free () ;

機 能 スタック領域の残りバイト数を調べます。

戻 り 値 スタック領域の残りバイト数を返します。

b_seek

レベル 3

書 式 #include <basic.h>

```
int b_seek(fp, ofs, md) ;  
int fp; /* ファイルポインタを移動するファイルのファイル番号 */  
int ofs; /* オフセット /* 開始位置から移動するバイト数  
int md; /* ファイルポインタの起点 */
```

機 能 fpで指定したファイルのファイルポインタを、mdを起点としてofs バイト数分移動します。
md は、次の意味をもちます。

- 0 : 先頭を起点とする
- 1 : 現在位置を起点とする
- 2 : 最後を起点とする

戻 り 値 先頭からのオフセット値を返します。
オープンしていないファイルを指定したときは、-2を返します。
その他のエラーのときには、-1を返します。

b_write

レベル 3

書 式 #include <basic.h>

```
int b_write(buff, size, count, fp) ;  
void *buff; /* 書き出し領域へのポインタ */  
int size; /* 書き出す項目のサイズ */  
int count; /* 書き出す項目数 */  
int fp; /* 該当ファイルのファイル番号 */
```

機 能 buff が指すバッファから、size×count バイト数分、fp で指定したファイルヘッダータを書き出します。

戻 り 値 実際に書き出した項目数を返します。

オープンしていないファイルに書こうとしたときは、-2 を返します。
その他のエラーのときには、-1 を返します。

b_fwrites

レベル 3

書 式 #include <basic.h>

```
int b_fwrites(buff, fp) ;  
char *buff; /* 書き出し領域へのポインタ */  
int fp; /* 該当ファイルのファイル番号 */
```

機 能 バッファ buff に入っている文字列をファイルに書き出します。

戻 り 値 実際に書き込んだ文字数を返します。
オープンしていないファイルに書こうとしたときは、-2 を返します。
その他のエラーのときには、-1 を返します。

参 照 関 数 b_freads, b_fopen, b_fwrite

bg_fill

レベル 3

書 式 #include <sprite.h>

```
int bg_fill(t_page, pattern) ;
char t_page; /* テキストページ */
int pattern; /* データパターン */
```

機 能 t_pageで指定したバックグラウンドテキストページを、patternで指定したデータで埋めつくします。
t_pageにはテキストページを指定します (0~1)。
patternに指定する内容は次の通りです。

ビット 31~16 : 常に 0
ビット 15 : 垂直反転
ビット 14 : 水平反転
ビット 12~13 : 常に 0
ビット 11~8 : パレットブロック
ビット 7~0 : PCG パターンコード

戻 り 値 以下に示す終了コードを返します。

0 = 正常終了
-1 = 画面モードエラー (画面サイズが768×512のときなど)

bg_get

レベル 3

書 式 #include <sprite.h>

```
long bg_get(t_page, xpos, ypos) ;
char t_page; /* テキストページ */
char xpos; /* X座標 */
char ypos; /* Y座標 */
```

機 能

バックグラウンドテキストを読み出します。
 t_page にはテキストページを指定します (0~1)。
 xpos には X 座標、ypos には Y 座標を指定します (いずれも 0~63)。

戻 り 値

バックグラウンドテキストのコードを返します。
 返される内容は次の通りです。

ビット 31~16 : 常に 0
 ビット 15 : 垂直反転
 ビット 14 : 水平反転
 ビット 12~13 : 常に 0
 ビット 11~8 : パレットブロック
 ビット 7~0 : PCG パターンコード

ただし、-1 が返された場合は画面モードエラーです。
 (画面サイズが 768×512 のときなど)

参 照 関 数 bg_put

bg_put

レベル 3

書 式 #include <sprite.h>

```
int bg_put(t_page, xpos, ypos, pattern);
char t_page; /* テキストページ */
char xpos; /* X座標 */
char ypos; /* Y座標 */
int pattern; /* データパターン */
```

機 能 バックグラウンドテキストに書き込みます。

t_page にテキストページを指定します (0~1)。

xpos には X 座標、ypos には Y 座標を指定します (いずれも 0~63)。

pattern にはデータパターンを指定します。

pattern に指定する内容は次の通りです。

ビット 31~16 : 常に 0

ビット 15 : 垂直反転

ビット 14 : 水平反転

ビット 13~12 : 常に 0

ビット 11~8 : パレットブロック

ビット 7~0 : PCG パターンコード

戻 り 値 以下に示す終了コードを返します。

0 = 正常終了

-1 = 画面モードエラー (画面サイズが 768×512 のときなど)

参 照 関 数 bg_get

bg_scroll

レベル 3

書 式 #include <sprite.h>

```
int bg_scroll(bg_no, x, y);  
char bg_no; /* バックグラウンド番号 */  
int x; /* スクロール X 座標 */  
int y; /* スクロール Y 座標 */
```

機 能 bg_no で指定するバックグラウンド (0~1) に、x, y で示すスクロール座標を設定します。

戻 り 値 以下に示す終了コードを返します。

- 0 = 正常終了
- 1 = 画面モードエラー (画面サイズが 768×512 のときなど)

bg_set

レベル 3

書 式 #include <sprite.h>

```
int bg_set(bg_no, t_page, disp) ;
char bg_no; /* バックグラウンド番号 */
int t_page; /* テキストページ */
int disp; /* 表示モード */
```

機 能

バックグラウンドテキストの設定を行います。
 bg_no にはバックグラウンド番号を指定します (0~1)。
 t_page にはテキストページを指定します (0~1)。
 disp には表示モードを指定します。
 1 を指定した場合には表示を行い、0 を指定した場合には表示は行いません。

戻 り 値

以下に示す終了コードを返します。

- 0 = 正常終了
- 1 = 画面モードエラー (画面サイズが 768×512 のときなど)

bg_stat

レベル 3

書 式 #include <sprite.h>

```
int bg_stat(bg_no, sel_mode) ;
char bg_no; /* バックグラウンド番号 */
char sel_mode; /* モード */
```

機 能 指定したバックグラウンドの状態を調べます。

bg_no にはバックグラウンド番号を指定します (0~1)。

sel_mode には調べたい内容を指定します。

0 : X 座標

1 : Y 座標

2 : テキストページ (0~1)

3 : 表示状態 (0=表示していない、1=表示している)

戻 り 値

sel_mode に合わせて、それぞれの値を返します。

ただし、-1 が返された場合は画面モードエラーです。

(画面サイズが 768×512 のときなど)

b_hexS

レベル 3

書 式 #include <basic.h>

```
char *b_hexS (s, n) ;
char *s; /* 文字列格納領域へのポインタ */
int n; /* 数値 */
```

機 能 数値 n を 16 進数表現の文字列に変換し、s が指す領域へ格納します。
s が指す領域は、最低 10 バイト以上なければなりません。

戻 り 値 変換結果の文字列へのポインタを返します。

b_init

レベル 3

書 式 #include <basic0. h>

void b_init();

機 能 BASIC から C へ変換したときに BASIC と同じ状態で動作するための初期化を行います。

初期化の内容は次の通りです。

- 現在の画面モードの退避
- ファンクションキーの内容の退避
- BASIC 用ファンクションキーの設定
- アボート処理の登録
- 乱数の初期化
- カーソルの非表示

戻 り 値 戻り値はありません。

b_inkey0

レベル 3

書 式 #include <basic0. h>

```
char *b_inkey0(s) ;  
char *s; /* キーコード格納領域へのポインタ*/
```

機 能 キー入力待ちを行わずにキー入力を行い、入力した文字を s が指す領域に格納します。
入力した文字には '¥0' が付加されますので、s が指す領域は 2 バイト以上必要です。

戻 り 値 入力した文字へのポインタを返します。

b_inkeyS

レベル 3

書 式 #include <basic0. h>

```
char *b_inkeyS(s) ;  
char *s; /* キーコード格納領域へのポインタ */
```

機 能 キーの入力を待ってキー入力を行い、入力した文字を s が指す領域に格納します。
入力した文字には '¥0' が付加されますので、s が指す領域は 2 バイト以上必要です。

戻 り 値 入力した文字へのポインタを返します。

b_input

レベル 3

書 式 `#include <basic0.h>`

```
void b_input (prmptr, [type, varadr] . . . , -1) ;
char *prmptr; /* プロンプト文字列へのポインタ */
int type; /* 変換書式 */
void *varadr; /* 読み込み領域のアドレス */
```

機 能 prmptr が指すプロンプト文字列を表示して入力を待ち、type に従って変換し、varadr から始まる領域に値を格納します。

文字列の場合は、type で示す長さ-1 までが有効な入力となります。

入力する変数が複数個の場合には入力時に“,”で区切ります。

入力データ中に“,”や“ ””を使うことはできません。

type には、次の値を指定します。

- 1 : 終りを表します (varadr は指定しません)
- 1~257 : 文字列の長さ (文字列の最後の¥0 を含む長さ)
- 0x201 : 文字型 (unsigned char)
- 0x204 : 整数型 (int)
- 0x208 : 浮動小数点型 (double)

type と varadr の型は、以下のように対応していなければなりません。

- | type | varadr の型 |
|-------|-------------------|
| 1~257 | : unsigned char * |
| 0x201 | : unsigned char * |
| 0x204 | : int * |
| 0x208 | : double * |

戻 り 値 戻り値はありません。

b_int

レベル 3

書 式 #include <basic0.h>

```
int b_int(f);  
double f; /* 変換される浮動小数点数 */
```

機 能 浮動小数点数を整数型に変換します。

戻 値 変換した数値を返します。

b_iprint

レベル 3

書 式 #include <basic0. h>

void b_iprint(val) ;

int val; /* 整数値 */

機 能 BASIC から C に変換したときに PRINT 文の引数が整数のとき、b_iprint に展開されます。

整数を画面に出力します。

戻 り 値 戻り値はありません。

b_itoa

レベル 3

書 式

```
#include <basic.h>
```

```
char *b_itoa(str, val) ;
```

```
char *str; /* 文字列格納領域へのポインタ */
```

```
int val; /* 整数値 */
```

機 能

BASIC から C に変換したときに ITOA 関数は、b_itoa に展開されます。

数値 val を文字列に変換し、str が指す領域へ格納します。

str が指す領域には、最低 13 バイト以上の領域が必要です。

戻 り 値

変換結果の文字列へのポインタを返します。

bldmem, allmem

レベル 2

書 式 #include <stdlib.h>

```
int bldmem(n) ; /* メモリブロックの確保 */
int n; /* 確保する領域のサイズ (キロバイト単位) */

int allmem( ) ; /* 使用可能な全メモリ領域の確保 */
```

機 能

メモリ領域の確保を行います。

bldmem は、メモリブロックをキロバイト単位で確保します。

n に 0 を指定すると、使用可能な全メモリ領域を確保します。

allmem は、使用可能な全メモリ領域の確保を行います。

これらの関数によって確保した領域は、getmem 関数などを用いて割りあて、使用することができます。

戻 り 値

処理が正常に終了した場合には 0 を返し、領域が確保できないときには -1 を返します。

プログラム例

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

main()
{
    int n;

    if (allmem() == 0)
        printf("全メモリを確保しました。 %n");
    else {
        printf("メモリの確保に失敗しました。 %n");
        exit(1);
    }
    rstmem();
    printf("すべてのメモリを解放しました %n");

    for (n = 0; n < 1000; n++) {
        printf("%dKbyte: ", n);
        if (bldmem(n) == 0) {
            printf("確保できました。");
            rstmem();
            printf(": 全メモリを解放しました %n");
        }
        else {
            printf("確保できませんでした。 %n");
            break;
        }
    }
}
```

b_leftS

レベル 3

書 式 #include <basic.h>

```
char *b_leftS(s, sl, i);
char *s; /* 結果文字列格納領域へのポインタ */
char *sl; /* 文字列 */
int i; /* 文字数 */
```

機 能 文字列 sl の左端から i 文字分とり出し、s が指す領域へ格納します。
s が指す領域は、最低 256 バイト以上なければなりません。
また、sl が指す文字列の長さは、255 以下でなければなりません。

戻 り 値 とり出した文字列 s へのポインタを返します。

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

int main()
{
    int n;
    if (argc == 0)
        printf("文字列と文字数を指定してください。");
    else {
        printf("文字列と文字数を指定してください。");
        exit(1);
    }
    printf("文字列と文字数を指定してください。");
    for (n = 0; n < 1000; n++) {
        printf("XbByte: ", n);
        if (b_leftS(n) == 0)
            printf("文字列と文字数を指定してください。");
        else {
            printf("文字列と文字数を指定してください。");
            break;
        }
    }
}
```


b_input

レベル 3

書 式 #include <basic0. h>

```
void b_input(s, size) ;  
char *s; /* 入力文字列の格納領域へのポインタ */  
int size; /* 入力する領域のバイト数 */
```

機 能 キーボードからの入力を、最大 size-1 文字数、s が指す領域に格納します。
size には、文字列の終りを示す '¥0' を含めた領域のバイト数を指定します。
文字列の最後には '¥0' が付加されます。
したがって、s が指す領域には最大で size-1 の文字が入力されます。
ただし、2 行にわたっての入力はできません。

戻 り 値 戻り値はありません。

b_midS

レベル 3

書 式 #include <basic.h>

```
char *b_midS(s, s1, bgn, count) ;  
char *s; /* 結果文字列格納領域へのポインタ */  
char *s1; /* 文字列 */  
int bgn; /* 開始位置 */  
int count; /* 文字数 */
```

機 能 文字列 s1 の bgn 文字目から count 文字だけを取り出し、s が指す領域へ格納します。
s が指す領域は、最低 256 バイト以上なければなりません。
また、s1 が指す文字列の長さは、255 以下でなければなりません。

戻 り 値 とり出した文字列 s へのポインタを返します。

b_mirrorS

レベル 3

書 式 #include <basic.h>

```
char *b_mirrorS(s, s1) ;  
char *s; /* 結果文字列格納領域へのポインタ */  
char *s1; /* 文字列 */
```

機 能 文字列 s1 の並びを前後逆にして、s が指す領域へ格納します。
s が指す領域は、最低 256 以上バイトなければなりません。
また、s1 が指す文字列の長さは、255 以下でなければなりません。

戻 り 値 並びかえた文字列 s へのポインタを返します。

b_octS

レベル 3

書 式 #include <basic.h>

```
char *b_octS(s, n) ;  
char *s; /* 文字列格納領域へのポインタ */  
int n; /* 数値 */
```

機 能 数値 *n* を 8 進数表現の文字列に変換し、*s* が指す領域へ格納します。
s が指す領域は最低 13 バイト以上なければなりません。

戻 り 値 変換結果の文字列へのポインタを返します。

box

レベル 3

書 式 #include <graph.h>

関数呼び出し

```
void box (x1, y1, x2, y2, p, ls) ;
int x1; /* 始点 X 座標 */
int y1; /* 始点 Y 座標 */
int x2; /* 終点 X 座標 */
int y2; /* 終点 Y 座標 */
int p; /* パレットコード */
int ls; /* ラインスタイル */
```

機 能 グラフィック画面に長方形を描画します。

x1 には始点 X 座標を指定します (−32768~32767)。

y1 には始点 Y 座標を指定します (−32768~32767)。

x2 には終点 X 座標を指定します (−32768~32767)。

y2 には終点 Y 座標を指定します (−32768~32767)。

p にはパレットコードを指定します (0~65535)。

画面モードにより最大値が異なります。

ls にはラインスタイルを指定します (0~65535)。

戻 り 値 戻り値はありません。

プログラム例

```

#include      <stdlib.h>
#include      <basic0.h>
#include      <graph.h>

main()
{
    int      i;
    int      x1, x2, y1, y2;
    int      pc;
    int      ls;

    b_init();
    screen(2, 0, 1, 1);
    vpage(1);
    for (i = 0; i < 500; i++) {
        x1 = rand() % 768;
        y1 = rand() % 512;
        x2 = rand() % 768;
        y2 = rand() % 512;
        pc = rand() % 16;
        ls = rand() * 2 + 1;
        box(x1, y1, x2, y2, pc, ls);
    }
    b_exit(0);
}

```

b_pi

レベル 3

書 式 #include <basic0. h>

```
double b_pi(f) ;  
double f; /* 倍率 */
```

機 能 円周率 π を f 倍した値を求めます。

戻 り 値 $f \times \pi$ の値を返します。

参 照 関 数 pi

b_rightS

レベル 3

書 式 #include <basic.h>

```
char *b_rightS(s, sl, i) ;  
char *s; /* 結果文字列格納領域へのポインタ */  
char *sl; /* 文字列 */  
int i; /* 文字数 */
```

機 能 文字列 sl の右端から i 文字分とり出し、s が指す領域へ格納します。
s が指す領域は、最低 256 バイト以上なければなりません。
また、sl が指す文字列の長さは、255 以下でなければなりません。

戻 り 値 とり出した文字列 s へのポインタを返します。

bsearch

レベル 2

書 式 #include <stdlib.h>

関数で探す

```
void *bsearch(key, base, num, width, compare);
void *key; /* サーチするデータを指すポインタ */
void *base; /* 配列を指すポインタ */
size_t num, width; /* 配列要素の数、配列要素のサイズ */
int (*compare)(); /* compare ルーチンを指すポインタ */
```

機 能

各要素が width バイトで、要素数が num 個の配列をバイナリサーチします。

base は配列を指すポインタで key はサーチする値へのポインタです。

なお、この配列はあらかじめ昇順にソートされていなければなりません。

compare は、サーチする値と配列の要素を比較し、その大小関係を返すユーザーが用意するルーチンを指すポインタです。

bsearch は、サーチの過程で、必要に応じて compare が指す関数を呼び出します。

compare は 2 つの引数を受けとり、1 番目の引数はサーチする値へのポインタ、2 番目の引数は配列の要素へのポインタが渡されるので、その 2 つを比較して、次の値のうちいずれかを返す関数でなければなりません。

compare の戻り値	内 容
負	サーチする値が要素よりも小さい
0	サーチする値と要素は等しい
正	サーチする値が要素よりも大きい

戻 り 値

最初に見つかった配列要素へのポインタを返します。

見つからなかった場合は NULL を返します。

参 照 関 数

qsort

プログラム例

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

#define MAX 12

typedef struct {
    char *key;
    char *name1;
    char *name2;
} ELM;

/* This table must be sorted by alphabetical */
static ELM table[MAX] = {
    {"Aqr", "Aquarius", "水瓶"},
    {"Ari", "Aries", "牡羊"},
    {"Cap", "Capricornus", "山羊"},
    {"Cnc", "Cancer", "蟹"},
    {"Gem", "Gemini", "双子"},
    {"Leo", "Leo", "獅子"},
    {"Lib", "Libra", "天秤"},
    {"Psc", "Pisces", "魚"},
    {"Sco", "Scorpius", "蠍"},
    {"Sgr", "Sagittarius", "射手"},
    {"Tau", "Taurus", "牡牛"},
    {"Vir", "Virgo", "乙女"}
};

main(int argc, char *argv[])
{
    int compare(char **, ELM *);
    char *key;
    ELM *p;

    if (argc < 2)
        exit(1);
    key = argv[1];
    if ((p = (ELM *)bsearch(&key, (char *)table,
        MAX, sizeof(ELM), compare)) != NULL)
        printf("not found\n");
    else
        printf("%s: %s\n", p->name1, p->name2);
}

int compare(char **key, ELM *elm)
{
    return (strcmp(*key, elm->key));
}
```

b_setdate

レベル 3

書 式 #include <basic0. h>

```
void b_setdate(s) ;  
char *s; /* 日付を表す文字列へのポインタ */
```

機 能 システムクロックの日付を設定します。
sが指す文字列は "87/09/30" のような形式で指定してください。

戻 り 値 戻り値はありません。

b_settime

レベル 3

書 式 #include <basic0. h>

```
void b_settime(s) ;  
char *s; /* 時刻を表す文字列へのポインタ */
```

機 能 システムクロックの時刻を設定します。
s が指す文字列は "21:56:30" のような形式で指定してください。

戻 り 値 戻り値はありません。

b_spaceS

レベル 3

書 式 #include <basic0. h>

```
char *b_spaceS(s, n) ;  
char *s; /* 作成した文字列を格納する領域へのポインタ */  
int n; /* 文字数 */
```

機 能 s の指す領域に、n 文字の空白文字からなる文字列を作成します。
n に 256 以上の値を指定すると、n は 0 とみなされます。
n に 0 を指定すると、長さが 0 の文字列（空文字列）を作成します。
s が指す領域は、最低 256 バイト以上なければなりません。

戻 り 値 作成した文字列 s へのポインタを返します。

b_sprint

レベル 3

書 式 #include <basic0. h>

```
void b_sprint(s) ;  
char *s; /* 表示文字列へのポインタ */
```

機 能 sが指す文字列をテキスト画面に表示します。

戻 り 値 戻り値はありません。

b_stradd

レベル 3

書 式 #include <basic0.h>

```
char *b_stradd(s, s1, s2, . . . , -1) ;
char *s; /* 連結した文字列を格納する領域へのポインタ */
char *s1; /* 文字列へのポインタ */
char *s2; /* 文字列へのポインタ */
```

機 能

複数の文字列を連結した文字列を作ります。

連結した文字列の長さが 255 文字以上になる場合は、255 文字まで連結し、それ以降の文字は連結されません。

s が指す領域は、最低 256 バイト以上なければなりません。

引き数の最後には、これ以上連結する文字列がないことを表す -1 を指定します。

戻 り 値

連結した文字列へのポインタを返します。

b_strchr, b_strrchr, b_strtok

レベル 3

書 式

```
#include <basic.h>
```

```
int b_strchr(str, ch) ;  
int b_strrchr(str, ch) ;  
char *b_strtok(str1, str2) ;  
  
char *str, *str1, *str2;  
char ch;
```

機 能

b_strchr は、str の中の最初にみつかった ch の位置を求めます。

b_strrchr は、str の中の最後にみつかった ch の位置を求めます。

b_strtok は検索文字集合 str2 に含まれる文字のいずれか 1 つの文字を str1 から探し、みつかった位置までの文字列を求めます。

b_strtok の 2 回目以降の呼び出しで、str1 に長さ 0 の文字列を指定すると、前回の続きから検索します。

このとき、検索文字集合は前回と異なる文字集合を指定してもかまいません。

BASIC の strchr, strrchr, strtok がそれぞれ展開されたもので、C の標準ライブラリにある strchr, strrchr, strtok とは仕様が異なります。

b_strtok は、内部の静的な領域へのポインタを返しますので、返されたポインタが指す領域を書き変えてはいけません。

また、str1 に長さが 0 でない文字列を指定すると、以前に返された文字列は失われます。

戻 り 値

b_strchr, b_strrchr は、ch の見つかった位置を返します。

見つからないときは -1 を返します。

b_strtok は、見つかった文字列へのポインタを返します。

見つからないときは、長さが 0 の文字列へのポインタを返します。

b_strcmp, b_strncpy

レベル 3

書式

#include <basic0.h>

```
int b_strcmp(str1, op, str2) ;
void b_strncpy(num, str1, str2) ;

char *str1, *str2 ;
int op, num ;
```

機能

b_strcmp は、2つの文字列 str1, str2 を比較し、指定された比較条件が成り立つかどうか調べます。

BASIC の文字列比較 (if a\$ < "abc" など) が展開されたもので、C のライブラリにある strcmp とは仕様が異なります。

<= や < などの記号は、op にわたされます。

OP に指定できる条件は以下のとおりです。

'=' (0x3d20)	等しい
'==' (0x3d3d)	等しい
'<>' (0x3c3e)	等しくない
'><' (0x3e3c)	等しくない
'!=' (0x213d)	等しくない
'>=' (0x3e3d)	大きいか等しい
'=>' (0x3c3e)	大きいか等しい
'<=' (0x3c3d)	小さいか等しい
'=<' (0x3d3c)	小さいか等しい
'<' (0x3c20)	小さい
'>' (0x3e20)	大きい

b_strncpy は、コピーする文字数+1 を num に指定して、文字列 str2 を文字列 str1 にコピーします。

str1 が指す領域は、num で示すバイト数の領域が必要です。

num には、文字列の終りを示す '¥0' を含めた文字数を指定するので、通常は sizeof (str1) のように指定します。

num に 0 を指定すると、str1 の文字列は変化しません。

また、1 を指定すると、str1 の文字列の長さが 0 になります。

BASIC の文字列代入 (a\$="abc" など) が展開されたもので、C の標準ライブラリにある strncpy とは仕様が異なります。

3 (ハレ)

戻り値

b_strcmp は、比較の結果、op に指定された条件が成立した場合は 0 以外を返し、不成立の場合は 0 を返します。

結果	戻り値
成り立つ	0 以外
成り立たない	0

b_strncpy は、戻り値はありません。

b_strfS

レベル 3

書 式 #include <basic0. h>

```
char *b_strfS(s, f) ;  
char *s; /* 文字列格納領域へのポインタ */  
double f; /* 浮動小数点数値 */
```

機 能 浮動小数点値 f を文字列に変換し、s が指す領域に格納します。
変換する数値が負の数の中には、負の符号がつきます。
s が指す領域には、最低 23 バイト以上の領域が必要です。

戻 り 値 変換した文字列 s へのポインタを返します。

b_striS

レベル 3

書 式 #include <basic0. h>

```
char *b_striS(s, n) ;  
char *s; /* 文字列格納領域へのポインタ */  
int n; /* 整数値 */
```

機 能 整数値 *n* を文字列に変換し、*s* が指す領域に格納します。
変換する数値が負の数の中には、負の符号がつきます。
s が指す領域には、最低 13 バイト以上の領域が必要です。

戻 り 値 変換した文字列 *s* へのポインタを返します。

b_stringS

レベル 3

書 式	#include <basic.h>
-----	--------------------

```
char *b_stringS(s, i, st) ;
char *s; /* 文字列格納領域へのポインタ */
int i; /* 並べる文字数 */
char *st; /* 該当文字格納領域へのポインタ */
```

機 能	文字列 st で指定した文字列の先頭 1 文字を、i 文字分並べた文字列を作成し、s が指す領域へ格納します。 BASIC の string\$ が展開されたものです。 i に 256 以上を指定すると、i は 0 とみなされます。 i に 0 を指定すると、長さが 0 の文字列（空文字列）を作成します。 s が指す領域は少なくとも 256 バイトなければなりません。
-----	---

戻 り 値	作成した文字列 s へのポインタを返します。
-------	------------------------

b_timeS

レベル 3

書 式 #include <basic0. h>

```
char *b_timeS(s) ;  
char *s; /* 時刻設定文字列へのポインタ */
```

機 能 システムクロックの時刻を“tt:mm:ss”の形式の文字列にして、sが指す領域に格納します。

戻 り 値 作成した文字列へのポインタを返します。

b_tpalet

レベル 3

書 式 #include <basic0. h>

```
void b_tpalet(p0, p1, p2, p3) ;
int p0, p1, p2, p3; /* パレット属性 */
```

機 能 テキストのパレット色を設定します。
p0～p3 にはパレットコード 0～3 に対する色コードを指定します。

戻 り 値 戻り値はありません。

b_tprint

レベル 3

書 式 #include <basic0. h>

```
void b_tprint(n) ;  
int n; /* タブスキップ数 */
```

機 能 カーソルの桁の位置が n より小さければカーソル位置が n になるまでスペースを表示します。
n より大きいとか等しければ何も行いません。

戻 り 値 戻り値はありません。

cabs

レベル 2

書 式 `#include <math.h>`

```
double cabs(z);
struct complex *z;
```

機 能 複素数の絶対値を求めます。

`z` は、複素数を格納してある `complex` 型構造体へのポインタを指定します。

```
struct complex {
    double x, y;
};
```

また、`cabs` は下の式と同じ結果を与えます。

```
sqrt (z->x*z->x+z->y*z->y)
```

戻 り 値 複素数の絶対値を返します。

参 照 関 数 `abs`, `fabs`, `labs`

calloc

レベル 2

書 式 `#include <stdlib.h>`

```
void    *calloc(n, size) ;  
size_t n; /* 配列要素の個数 */  
size_t size; /* 配列要素のサイズ */
```

機 能 各要素の大きさが size バイトで、要素数が n 個の配列のための領域を割りつけ
ます。
確保された配列は 0 で初期化されます。

戻 り 値 割りつけた領域を指すポインタを返します。
領域が確保できなかったときには、NULL を返します。

参 照 関 数 free, malloc, realloc

ceil

レベル 2

書 式 `#include <math.h>`

```
double ceil(x) ;
double x ;
```

機 能 x より小さくない最小の整数を求めます。

戻 り 値 double としての整数を返します。

参 照 関 数 floor, modf

cgets

レベル 2

書 式 `#include <conio.h>`

```
char *cgets(str);  
char *str; /* 文字列格納領域へのポインタ */
```

機 能 コンソールから文字列を読み込み、その長さと文字列をを配列 str に格納します。

str が指す領域は、読み込まれる文字列の最大長+3 バイトなければなりません。

str [0] : 読み込まれる文字列の最大長
str [1] : 読み込まれた文字列の実際の長さ
str [2] : 格納した文字列の先頭位置

cgets は、復帰改行を読み込むか指定された文字数 str [0] に達するまでコンソールから入力される文字を読み続け、str [2] から順に格納します。

復帰改行を読み込むと、それをヌル文字 ('¥0') に置き換えて格納します。

読み込みが終了すると、str [1] に実際の文字列の長さを格納します。

戻 り 値 読み込んだ文字列へのポインタを返します。

参 照 関 数 fgets, gets

呼び出しの前に str [0] に読み込まれる文字列の最大長を代入しておかなければなりません。

chdir

レベル 2

書 式 #include <direct. h>

```
int chdir(pathname) ;
```

```
char *pathname; /* 変更先ディレクトリ名へのポインタ */
```

機 能 カレントディレクトリを pathname によって指定したディレクトリに変更します。

pathname が示すディレクトリは既存のものでなければなりません。

戻 り 値 正常終了ならば正の数 (0 も) を返します。
エラーならば -1 を返します。
また、エラーが発生した場合、errno と _doserrno にそれぞれ C のエラー番号と OS のエラー番号をセットします。

参 照 関 数 mkdir, rmdir

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

/* 関数定義 */
/* 関数定義 */
/* 関数定義 */
/* 関数定義 */
/* 関数定義 */
/* 関数定義 */
/* 関数定義 */
/* 関数定義 */
/* 関数定義 */
/* 関数定義 */

main()
{
    int r;
    r = chdir("C:\\WINDOWS\\SYSTEM32");
    if (r == 0)
        printf("成功しました。");
    else
        printf("失敗しました。");
}
```

chgfa

レベル 2

書 式 #include <io.h>

```
int chgfa(pathname, atr) ;
char *pathname; /* 属性を変更するファイルのパス名へのポインタ */
int atr; /* ファイルの属性 */
```

機 能 pathnameで指定したファイルの属性を、atrで指定した属性に変更します。
atrで指定できる属性は以下の通りです。

0x01	: 読み込み専用	}	組合せて 0x00~0x07 まで指定 できます。
0x02	: 隠しファイル		
0x04	: システムファイル		
0x08	: ボリューム名	}	3つの中から1つだけ指定し、 さらに上記の 0x00~0x07 との 加算した値を設定します。
0x10	: ディレクトリ		
0x20	: 通常ファイル		

戻 り 値 正常終了ならば0を返します。
エラーならば-1を返します。

プログラム例

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

#define A_RDO 0x01 /* 読み込み専用 */
#define A_HDN 0x02 /* 隠しファイル */
#define A_SYS 0x04 /* システムファイル */
#define A_VOL 0x08 /* Volumeラベル */
#define A_DIR 0x10 /* ディレクトリ */
#define A_FIL 0x20 /* 通常ファイル */

main()
{
    int r;

    r = chgfa("¥¥CONFIG.SYS", A_RDO|A_FIL);

    if (r == 0)
        puts("成功しました。");
    else
        puts("失敗しました。");
}
```

chgft

レベル 2

書 式 #include <io.h>

関数呼び出し

```
int chgft(file, datetime);
int file; /* 作成日時を変更するファイルのファイルハンドル */
int datetime; /* ファイルの作成日時 */
```

機 能

file で指定したファイルの作成日時を、datetime で指定した作成日時に変更します。

datetime のフォーマットは以下の通りです。

ビット 31~25 : 1980 年からの相対年数(0~99)
 ビット 24~21 : 月(1~12)
 ビット 20~16 : 日(1~31)
 ビット 15~11 : 時(0~23)
 ビット 10~5 : 分(0~59)
 ビット 4~0 : 秒/2(0~29)

戻 り 値

正常終了ならば 0 を返します。
 エラーならば -1 を返します。

プログラム例

```
#include <stdio.h>
#include <time.h>
#include <io.h>
#include <fcntl.h>
#include <stat.h>

main()
{
    struct tm      *mp;
    time_t      tp;
    long      datetime;
    int      fn;
    char      pkttime[6];

    tp = time(NULL);
    mp = localtime(&tp);
    pkttime[0] = mp->tm_year - 80;
    pkttime[1] = mp->tm_mon + 1;
    pkttime[2] = mp->tm_mday;
    pkttime[3] = mp->tm_hour;
    pkttime[4] = mp->tm_min;
    pkttime[5] = mp->tm_sec;
    datetime = ftime(pkttime);

    if ((fn = open("¥¥CONFIG.SYS", O_RDWR)) == -1)
        puts("ファイルがオープンできません。");
    else {
        if (chgft(fn, datetime) == 0)
            puts("成功しました。");
        else
            puts("失敗しました。");
        close(fn);
    }
}
```


child

レベル 3

書 式 #include <basic.h>

```
int child(pathname) ;
char *pathname; /* 実行するファイルのパス名へのポインタ */
```

機 能 pathname で指定した実行ファイルをメモリ上にロードし、子プロセスとして起動します。

戻 り 値 正常終了ならば子プロセスの終了コードを返します。
エラーならば負の数を返します。

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <basic.h>

main()
{
    if (child("foo.X") < 0) {
        puts("エラー");
        exit(1);
    }
    else
        exit(0);
}
```

chkml

レベル 2

書 式 #include <stdlib.h>

long chkml();

機 能 未使用メモリブロックの最大サイズをバイト数で求めます。

戻 り 値 メモリサイズがバイト数で返ります。
メモリに空きがない場合は0が返ります。

プログラム例

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

```
main()
{
    long    size;

    size = chkml();
    printf("最大のブロックは %ldバイト\n",size);
}
```

chmod

レベル 2

書 式 `#include <stat. h>`
`#include <io. h>`

```
int chmod(pathname, pmode) ;
char *pathname; /* モードを変更するファイルのパス名 */
int pmode; /* アクセスモード */
```

機 能 `pathname` で指定したファイルのアクセスモードを `pmode` で指定したアクセスモードに変更します。

`pmode` は、`stat. h` で定義されている定数 `S_IWRITE`、`S_IREAD` か、またはその2つを論理和した値を指定します。

他の値を指定すると、アクセスモードは変更しません。

引数 `pmode` に指定できる値とその内容を以下に示します。

pmode の値	内 容
<code>S_IWRITE</code>	書き込み可
<code>S_IREAD</code>	読み込み可
<code>S_IREAD S_IWRITE</code>	読み書きともに可

書き込み可でないファイルは、読み出し専用になります。

Human68k では、`S_IWRITE` モードと `S_IREAD | S_IWRITE` モードは同じです。

戻 り 値 アクセスモードを変更することができた場合は 0 を返します。
 エラーが起きた場合は -1 を返し、`errno` にエラー番号がセットされます。
`ENOENT` は指定したファイルを発見できなかったことを表しています。

参 照 関 数 `access`, `creat`, `fstat`, `open`, `stat`

chsize

レベル 2

書 式 #include <io.h>

```
int chsize(handle, size) ;  
int handle; /* 該当ファイルのファイルハンドル */  
long size; /* ファイルの変更サイズ */
```

機 能 handle で指定したファイルの大きさを size で指定した大きさに変更します
この場合、ファイルはオープンされており、かつ書き込み可でなければなりません。

ファイルを大きくした場合にはヌル文字 ('¥0') がファイルの最後に大きくした
サイズ分だけ追加されます。

また、ファイルを小さくした場合は、ファイルの最後から小さくした分のデータ
が削除されます。

戻 り 値 ファイルの大きさを変更できた場合は 0 を返します。
変更できなかった場合は -1 を返し、errno に以下の値がセットされます。

errno の値	内 容
EACCES	指定したファイルハンドルは読み出し専用
EBADF	ファイルハンドルが無効
ENOSPC	デバイスに空きがない

参 照 関 数 close, creat, open

circle

レベル 3

書 式 #include <graph.h>

関数で呼ぶ

```
void circle (x, y, r, p, s, e, hv) ;
int x; /* 中心の X 座標 */
int y; /* 中心の Y 座標 */
int r; /* 半径 */
int p; /* パレットコード */
int s; /* 開始角度 */
int e; /* 終了角度 */
int hv; /* 偏平率 */
```

機 能 グラフィック画面に円、円弧、描画します。

x には中心の X 座標を指定します (−32768~32767)。

y には中心の Y 座標を指定します (−32768~32767)。

r には半径を指定します (0~32767)。

p にはパレットコードを指定します (0~65535)。

画面モードにより最大値が異なります。

s には円の開始角度を指定します (−360~360)。

開始角度に負の数を指定すると中心点から開始点までの直線を描画します。

e には円の終了角度を指定します (−360~360)。

終了角度に負の数を指定すると中心点から終了点までの直線を描画します。

hv には偏平率を指定します (0~65535)。

0~255 で縦の比率が hv/256 になります。

257~65535 で、横の比率が 256/hv になります。

256 で真円になります。

戻 り 値 戻り値はありません。

プログラム例

```

#include          <stdlib.h>
#include          <basic0.h>
#include          <graph.h>

main()
{
    int    i;
    int    x, y;
    int    r;
    int    pc;
    int    st, et;
    int    hv;

    b_init();
    screen(2, 0, 1, 1);
    for (i = 0; i < 500; i++) {
        x = rand() % 768;
        y = rand() % 512;
        r = rand() % 200;
        pc = rand() % 16;
        st = 0;
        et = 360;
        hv = 256;
        circle(x, y, r, pc, st, et, hv);
    }
    wipe();
    for (i = 0; i < 500; i++) {
        x = rand() % 768;
        y = rand() % 512;
        r = rand() % 200;
        pc = rand() % 16;
        st = 60;
        et = 30;
        hv = 256;
        circle(x, y, r, pc, st, et, hv);
    }
    wipe();
    for (i = 0; i < 500; i++) {
        x = rand() % 768;
        y = rand() % 512;
        r = 150;
        pc = rand() % 16;
        st = 0;
        et = 360;
        hv = rand() % 256;
        circle(x, y, r, pc, st, et, hv);
    }
    b_exit(0);
}

```

clearerr, clrerr

レベル 2

書 式 #include <stdio.h>

```
void clearerr(stream);
```

```
void clrerr(stream);
```

```
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能 指定されたストリームのエラーインジケータおよびエンドオブファイル (EOF) インジケータをリセットします。

一度エラーインジケータがストリームにセットされると、clearerr または rewind 関数を呼ぶまでそのストリームに対する処理はエラーになります。

戻 り 値 戻り値はありません。

参照関数 feof, ferror

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N_MIN 1
#define N_MAX 1000000

int
main()
{
    clock_t start;
    clock_t end;
    double sec;
    long sum;
    long i;

    if ((start = clock()) == -1)
        printf("process start time not available.\n");
    exit(1);

    start = clock();
    sum = 0;
    for (i = N_MIN; i <= N_MAX; i++)
        sum += i;
    end = clock();

    sec = (double)(end - start) / (double)CLOCKS_PER_SEC;
    printf("N[%d]までの和: %ld\n", N_MAX, sum);
    printf("経過時間: %f秒\n", sec);
}
```

clock

レベル 2

書 式 #include <time.h>

clock_t clock();

機 能 プロセスの使用時間を得ます。

clock は、プロセスの起動からの経過時間を返します。

ある処理の最初と最後で clock を呼び出し、その2つの時点での戻り値の差を求める事で、その処理に要した時間を測定する事ができます。

その時間を秒単位で表すには、マクロ CLK_TCK または CLOCKS_PAR_SEC で割る必要があります。

戻 り 値 プロセスの起動時からの経過時間を返します。

ただし、プロセスの経過時間が得られない場合や、その値が表現可能な範囲にならない場合は、-1 を返します。

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N_MIN 1L
#define N_MAX 1000000L

main()
{
    clock_t start;
    clock_t end;
    double sec;
    long sum;
    long l;

    if ((start = clock()) == -1) {
        printf("process起動時からの時間が得られません。¥n");
        exit(1);
    }
    start = clock();
    sum = 0;
    for (l = N_MIN; l <= N_MAX; l++)
        sum += l;
    end = clock();

    sec = (double)(end - start) / (double)CLK_TCK;
    printf("%ldから%ldまでの和: %ld¥n", N_MIN, N_MAX, sum);
    printf("処理に要した時間: %g秒¥n", sec);
}
```


close

レベル 2

書 式 #include <io.h>

```
int close(handle) ;
int handle; /* クローズするファイルのファイルハンドル */
```

機 能 handleで指定されたファイルをクローズします。

戻 り 値 ファイルが正常にクローズされた場合は0を返します。
クローズされなかった場合は-1を返し、errnoにEBADFがセットされます。
これはhandleが正しくないことを示しています。

参 照 関 数 creat, open

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <io.h>
#include <stat.h>

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

    file = (argc > 1) ? argv[1] : "file";

    if ((fn = open(file, O_RDONLY)) == -1) {
        printf("%sを開けません。%n", file);
        exit(1);
    }
    printf("%sを開しました。%n", file);

    if (close(fn) == -1) {
        printf("%sをcloseできません。%n", file);
        exit(1);
    }
    printf("%sをcloseしました。%n", file);

    exit(0);
}
```

cls

レベル 3

書 式 #include <basic0. h>

void cls() ;

機 能 テキスト画面をクリアします

なお、b_sprint (“¥x0c”) を実行した場合も同じ結果が得られます。

戻 り 値 戻り値はありません。

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <io.h>
#include <fcntl.h>

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

    if (argc > 1) {
        file = argv[1];
    } else {
        file = "test.txt";
    }

    if ((fd = open(file, O_RDONLY)) < 0) {
        perror("開けられませんでした。");
        exit(1);
    }

    printf("開けられました。 %s\n", file);

    if (close(fd) < 0) {
        perror("閉じられませんでした。");
        exit(1);
    }

    printf("閉じられました。 %s\n", file);

    exit(0);
}

```

color

レベル 3

書 式 `#include <basic0. h>`

```
void color(n) ;
int n; /* 文字属性 */
```

機 能 テキスト画面の文字属性を設定します。
n には次の内容を指定します。

0 = 黒	8 = 黒
1 = 水色	9 = 水色のリバーズ
2 = 黄色	10 = 黄色のリバーズ
3 = 白	11 = 白のリバーズ
4 = 黒	12 = 黒
5 = 水色の強調	13 = 水色の強調・リバーズ
6 = 黄色の強調	14 = 黄色の強調・リバーズ
7 = 白の強調	15 = 白の強調・リバーズ

なお、テキストのパレットは `b_tpalet` 関数で指定します。

戻 り 値 戻り値はありません。

参 照 関 数 `b_tpalet`

console

レベル 3

書 式 #include <basic0. h>

```
void console(ys, yl, fkey) ;  
int ys; /* スクロール開始行 */  
int yl; /* スクロール行数 */  
int fkey; /* ファンクションキー */
```

機 能 スクロール行の範囲を ys 行から yl 行分とします。

本関数実行後、カーソルはホームポジションに移動します。

fkey はファンクションキー表示の on/off の指定です。

0 を指定すると表示なし、1 を指定すると表示ありになります。

戻 り 値 戻り値はありません。

contrast

レベル 3

書 式 `#include <graph.h>`

```
void contrast(i) ;
int i; /* コントラスト */
```

機 能 画面のコントラストを0から15の値で指定します。

戻 り 値 戻り値はありません。

プログラム例

```
#include <basic0.h>
#include <graph.h>

#define N_CONT 16

main()
{
    int i;
    int j;

    b_init();
    screen(2, 0, 1, 1);
    vpage(1);
    symbol(300, 200, "x68000", 2, 2, 2, 15, 0);
    for (i = 0; i < N_CONT; i++) {
        contrast(N_CONT - i - 1);
        for (j = 0; j < 5000; j++)
            ;
    }
    b_exit(0);
}
```

cos, cosh

レベル 2

書 式 #include <math.h>

```
double cos(x) ;
double cosh(x) ;
double x ;
```

機 能 cos は x のコサインを、cosh は x のハイパボリック (双曲線) コサインを計算します。

戻 り 値 cos は x のコサインを返します。
cosh は x のハイパボリックコサインを返します。
引数が不正な値の場合は errno に EDOM がセットされ返り値は不定になります。
オーバーフローが発生した場合は errno に ERANGE がセットされ返り値は符号が付いた HUGE_VAL になります。
またエラー処理は、matherr 関数を使って行うことができます。

参 照 関 数 acos, asin, atan, atan2, matherr, sin, sinh, tan, tanh

プログラム例

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

main()
{
    double x;
    int i;

    for (i = -32; i <= 32; i++) {
        x = (double)i / 10.0;
        printf("cos(%4.1f) = %g\n", x, cos(x));
    }
    for (i = -32; i <= 32; i++) {
        x = (double)i / 10.0;
        printf("cosh(%4.1f) = %g\n", x, cosh(x));
    }
}
```

cprintf

レベル 2

書 式 `#include <conio.h>`

```
int cprintf(format_string [, argument...]) ;
char *format_string; /* 書式文字列へのポインタ */
```

機 能 `format_string` で指定した書式に従って出力を変換しコンソールに出力します。
`format_string` に続く引数は書式に従って変換され出力されます。
`format_string` の書式と機能は `printf` 関数と同じです。
 詳細については、`printf` 関数を参照してください。
 なお、`cprintf` は `fprintf`、`printf` 関数と異なり、改行 (`\n`) を復帰改行 (`\r\n`) には変換しません。

戻 り 値 出力した文字数を返します。

参 照 関 数 `fprintf`、`printf`、`sprintf`

プログラム例

```
#include <conio.h>

main()
{
    cprintf("hello\r\n");
    cprintf("world\r\n");

    cprintf("hello\n");
    cprintf("world\n");

    cprintf("\r\n");

    cprintf("[%d]\r\n", 10);
    cprintf("[%s]\r\n", "string");
}
```

cputs

レベル 2

書 式 #include <conio.h>

```
int cputs(str) ;  
char *str; /* 出力文字列を指すポインタ */
```

機 能 文字列 str をコンソールに出力します。
ただし文字列の末尾はヌル文字 ('¥0') でなくてはなりません。
また改行文字 ('¥n') を、復帰改行 (¥r¥n) に変換しません。
正しく復帰改行したい場合は、¥r¥n として下さい。

戻 り 値 正常終了したときは、0 を返します。
エラーが発生した場合は、0 以外を返します。

参 照 関 数 putchar

プログラム例

```
#include <conio.h>  
  
main()  
{  
    cputs("hello¥r¥n");  
    cputs("world¥r¥n");  
  
    cputs("hello¥n");  
    cputs("world¥n");  
}
```


creat

レベル 2

書 式 `#include <io.h>`
 `#include <stat.h>`

```
int creat(pathname, pmode);
char *pathname; /* 作成するファイルのパス名へのポインタ */
int pmode; /* アクセスモード */
```

機 能 `pathname` で指定したファイルが存在しない場合は、新しいファイルを `pmode` で指定したアクセスモードで作成します。

`pathname` で指定したファイルがすでに存在し、アクセスモードが書き込み可の場合は、既存のファイルの内容を削除して、書き込み用にオープンします。

`pmode` は `stat.h` で定義されている定数 `S_IWRITE`、`S_IREAD` か、またはその2つを論理和した値を指定します。

`pmode` の値とその内容については `chmod` を参照してください。

戻 り 値 ファイルが正しく作成された場合は、作成したファイルのファイルハンドルを返します。

正しく作成されなかった場合は `-1` を返し、`errno` に以下の値がセットされます。

errno の値	内 容
EACCES	指定したファイルはすでに存在し、読み出し専用のファイル、またはディレクトリ
EMFILE	ファイルハンドルがない
ENOENT	パス名がみつからない

参 照 関 数 `chmod`, `close`, `open`

書 式 #include <image. h>

void crt(screen_sw) ;

char screen_sw; /* 切り替えスイッチ */

機 能

テレビ、ビデオ画面の切り替えを行います。

screen_sw に指定できる内容は次の通りです。

0 : テレビ、ビデオの映像表示

1 : コンピュータ画面の表示

2 : テレビ、ビデオ映像のコントラストを下げてスーパーインポーズ表示

3 : テレビ、ビデオ映像のコントラストを通常にしてスーパーインポーズ表示

本関数の実行前に、screen 関数により低解像度モード (512×512、または 256×256) にしておく必要があります。

なお、crt (3) を実行すると、テロップ機能によりコンピュータ画像やスーパーインポーズの画面をビデオに録画することができます。

戻り値 戻り値はありません。

screen_sw	0
screen_sw	1
screen_sw	2
screen_sw	3

関連関数

chmod, close, open

cscanf

レベル 2

書 式 `#include <conio. h>`

```
int cscanf(format_string [, argument...]) ;  
char *format_string; /* 制御文字列 */
```

機 能 コンソールからデータを読み込み、format_string の書式に従って変換し、後に続く argument の指すフィールドに代入します。
format_string の内容は scanf と同じです。
また引数 argument は、format_string の各変換指定に対応する型変数のポインタです。

戻 り 値 代入したフィールドの数を返します。
代入しなかったフィールドは含みません。
フィールドに1つも代入しなかった場合は0を返します。
読み込みの前に入力が終わってしまった場合は EOF を返します。

参 照 関 数 fscanf, scanf, sscanf

csrlin

レベル 3

書 式 #include <basic0. h>

int csrlin();

機 能 カーソルの現在位置の Y 座標を得ます。

戻 り 値 カーソルの Y 座標を返します。

参 照 関 数 pos

参 照 関 数

参 照 関 数

ctime

レベル 2

書 式 #include <time.h>

```
char *ctime(time);
time_t *time;
```

機 能 time_t 型の値の時刻 time を文字列に変換します。

得られる文字列の書式は asctime 関数と同じです。

ctime は内部の静的な領域へのポインタを返しますので、返されたポインタが指す領域を書き変えてはいけません。

また、前回の呼び出して返された文字列は失われます。

それ以外にも、ctime は内部で localtime 関数を呼び出すので、それ以前に呼び出された、localtime 関数で返された構造体の内容も失われます。

戻 り 値 変換した文字列へのポインタを返します。

参 照 関 数 asctime, ftime, gmtime, time

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

int
main()
{
    time_t time;
    time_t time2;
    double diff;
    int i;

    time = time(NULL);
    time2 = time(NULL);
    diff = difftime(time2, time);

    for (i = 0; i < 10; i++)
        printf("%d\n", diff);
}
```

difftime

レベル 2

書式 #include <time.h>

```
double difftime(time1, time2) ;  
time_t time1; /* 引かれる値 */  
time_t time2; /* 引く値 */
```

機能 time1 から time2 を引いた時間を double で返します。time_t 型の時間情報は、そのまま引き算する事ができないので、この difftime 関数を用いて時間差を求め、double で得ることができます。

戻り値 time1 から time2 を引いた時間を double で返します。

プログラム例

```
#include <stdio.h>  
#include <time.h>  
  
main()  
{  
    time_t time1;  
    time_t time2;  
    double diff;  
    int i;  
  
    time(&time2);  
    for (i = 0; i < 0x2ffff; i++)  
    ;  
    time(&time1);  
    diff = difftime(time1, time2);  
    printf("%g秒\n", diff);  
}
```

div

レベル 2

書 式 #include <stdlib.h>

```
div_t div(number, denom);
```

```
int number; /* 割られる値 */
```

```
int denom; /* 割る値 */
```

機 能

number を denom で割った商と余りを、div_t 型の構造体として返します。

div は符号付き整数の除算を行う場合に、処理系の違いによる結果の相違を解消するために用意されています。

div_t 構造体は、符号付き整数 2 つをメンバに持つ構造体で、stddef.h で以下のように型宣言されています。

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

戻 り 値

number を denom で割った商と余りを div_t 型の構造体として返します。

戻り値は、構造体の実体である事に注意して下さい。

構造体のメンバ quot には商が、メンバ rem には余りが返ります。

参 照 関 数

ldiv

プログラム例

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

main()
{
    div_t  ans;
    int    num;
    int    denom;

    for (num = -10; num <= 10; num++) {
        for (denom = -10; denom <= 10; denom++) {
            if (denom != 0) {
                ans = div(num, denom);
                printf("%d / %d = %d 余り %d\n",
                    num, denom, ans.quot, ans.rem);
            }
        }
    }
}
```

dqsort, fqsort, lqsort, sqsort, tqsort

レベル 2

書 式 #include <stdlib.h>

```
void dqsort(da, n);
double *da; /* ソートする配列へのポインタ (double 型) */
int n; /* 配列の要素数 */
```

```
void fqsort(fa, n);
float *fa; /* ソートする配列へのポインタ (float 型) */
int n; /* 配列の要素数 */
```

```
void lqsort(la, n);
long *la; /* ソートする配列へのポインタ (long 型) */
int n; /* 配列の要素数 */
```

```
void sqsort(sa, n);
short *sa; /* ソートする配列へのポインタ (short 型) */
int n; /* 配列の要素数 */
```

```
void tqsort(ta, n);
char **ta; /* ソートする配列へのポインタ (ポインタ型) */
int n; /* 配列の要素数 */
```

機 能 ソートを行います。

dqsort は double 型、fqsort は float 型、lqsort は long 型、sqsort は short 型、tqsort はポインタ型の配列をそれぞれソートします。

注：この他に文字列をソートする関数として、strsrt があります。

strsrt の機能は tqsort と同じですが、バブルソートアルゴリズムを用いているので、サイズの小さい配列に対して効果的です。

戻 り 値 戻り値はありません。

参 照 関 数 strsrt

dskf

レベル 3

書 式 #include <basic.h>

```
int dskf(drive_no) ;
int drive_no; /* ドライブ番号 */
```

機 能 drive_no で指定したドライブのディスクの空き容量を調べます。
drive_no の指定できる内容は次の通りです。

```
0 : カレントドライブ
1 : ドライブ A
2 : ドライブ B
...
```

戻 り 値 ディスクの空き容量をバイト数で返します。

書 式 #include <io. h>

```
char *dtasc(date) ;  
int date; /* 文字列に変換したい日付情報 */
```

機 能

date で指定された日付情報を文字列に変換します。
date の内容は以下の通りです。

ビット 15～9 : 1980 年からの相対年数 (0～99)
ビット 8～5 : 月 (1～12)
ビット 4～0 : 日 (1～31)

date の形式は、dtnv 関数の戻り値と同じ形式です。

得られる文字列は 11 文字で、フォーマットは以下のようになります。

86-09-29¥0

dtasc は、内部の静的な領域へのポインタを返しますので、返されたポインタが指す領域を書き変えてはいけません。

また、前回の呼び出して返された文字列は失われます。

戻 り 値

正常終了ならば、変換した文字列へのポインタを返します。
エラーならば NULL を返します。

参 照 関 数

dtnv

プログラム例

```

#include      <stdio.h>
#include      <io.h>
#include      <fcntl.h>
#include      <stat.h>

main()
{
    char      *p;
    int       fn;
    long      datetime;

    fn = open("%%CONFIG.SYS", O_RDWR);
    datetime = getft(fn);

    if ((p = dtasc(datetime >> 16)) != NULL);
        printf("日付: %s\n", p);
    if ((p = tasc(datetime & 0xffff)) != NULL);
        printf("時刻: %s\n", p);
    close(fn);
}

```

文 章

説 明

変 数

関 数

マクロ

```

#include
#include
#include
#include

```

main()

```

int
int datetime;
int date;
int time;

```

```

if (date = dtasc(datetime >> 16)) < 0)
    printf("エラー\n");
if (time = tasc(datetime & 0xffff)) < 0)
    printf("エラー\n");

```

```

datetime = ((date << 16) | time);

```

```

fn = open("%%CONFIG.SYS", O_RDWR);
getft(fn, datetime);
close(fn);

```

書 式 #include <io.h>

```
int dtcnv(datestr);
char *datestr; /* 変換したい日付文字列 */
```

機 能 datestr で指定された文字列を日付情報に変換して、その値を返します。
指定する文字列は 11 文字で、フォーマットは以下のように指定します。

86-09-29¥0

datestr のフォーマットは、dtasc 関数の戻り値と同じフォーマットです。

戻 り 値 正常終了ならば、以下に示す形式で日付情報を返します。

ビット 15~9: 1980 年からの相対年数 (0~99)

ビット 8~5 : 月 (1~12)

ビット 4~0 : 日 (1~31)

エラーならば、負の数を返します。

参 照 関 数 dtasc

プログラム例

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <stat.h>

main()
{
    int fn;
    int datetime;
    int date;
    int time;

    if ((date = dtcnv("90-03-14")) < 0)
        printf("エラー¥n");
    if ((time = tmcnv("12:30:00")) < 0)
        printf("エラー¥n");

    datetime = ((date << 16) | time);

    fn = open("¥¥CONFIG.SYS", O_RDWR);
    chgft(fn, datetime);
    close(fn);
}
```

dup, dup2

レベル 2

書 式 #include <io. h>

```
int dup(handle) ;
int handle; /* ファイルハンドル */

int dup2(handle1, handle2) ;
int handle1; /* ファイルハンドル */
int handle2; /* 新たに割りあてられるファイルハンドル */
```

機 能

現在オープンされているファイルのファイルハンドルをコピーして、新しいファイルハンドルを作成します。

もとのファイルハンドルと新しいファイルハンドルは、同じファイルポインタを共有しますので、どちらのファイルハンドルを使ってもかまいません。

またファイルのアクセスモードは変化しません。

dup2 は、handle2 が handle1 と同じファイルを参照するようにします。

handle2 がオープンされているときは、まずクローズにした後に行います。

戻 り 値

dup の場合、新しいファイルハンドルを返します。

dup2 の場合は、正の数 (0 も) で正常終了、-1 でエラーです。

正確には、複写先のファイルハンドラ (2 番目のパラメータの内容) を返します。

参 照 関 数

close, creat, open

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <stat.h>
#include <io.h>

main()
{
    int    fn1;
    int    fn2;

    if ((fn1 = open("dup.c", O_TEXT|O_RDWR)) == -1) {
        printf("open失敗\n");
        exit(1);
    }
    printf("open %d\n", fn1);
    if ((fn2 = dup(fn1)) == -1) {
        printf("dup()失敗\n");
        exit(1);
    }
    printf("dup %d\n", fn1);
    if ((fn2 = dup2(2, fn1)) == -1) {
        printf("dup2()失敗\n");
        exit(1);
    }
    printf("dup2 %d\n", fn2);
}
```

ecvt

レベル 2

書 式 `#include <stdlib.h>`

```
char *ecvt(value, ndigits, decptr, signptr) ;  
double value; /* 変換すべき浮動小数点値 */  
int ndigits; /* 文字列の桁数 */  
int *decptr; /* 小数点位置を返すポインタ */  
int *signptr; /* 符号を返すポインタ */
```

機 能 浮動小数点値 `value` を、`ndigits` で指定した桁数の文字列に変換します。

生成された文字列の末尾にヌル文字‘` `’をつけ加えます。

`value` の桁数が `ndigits` の桁数より大きい場合は四捨五入され、小さい場合は文字列のうしろに 0 をつけます。

`decptr` は、小数点の位置を示す整数値で、0 または負の数字の場合は小数点が左端にあることを示します。

`signptr` は、数値の符号を示す整数値で、0 の場合は正の数、0 以外の場合は負の数であることを示します。

文字列に変換できなかった時は、長さが 0 の文字列にします。

`ecvt` は、内部の静的な領域へのポインタを返しますので返されたポインタが指す領域を書き変えてはいけません。

また、前回の呼び出して返された文字列は失われます。

なお、`printf` などでも、`%e`、`%E` の書式指定を使用すると、`ecvt` が呼び出されるので、それ以前に `ecvt` で返された文字列は失われます。

戻 り 値 変換された文字列へのポインタを返します。

参 照 関 数 `atof`, `fcvt`, `gcvt`

プログラム例

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

#define PI 3.1415926535897932385

main()
{
    int dec;
    int sign;
    char *buf;
    int n;

    for (n = 1; n <= 100; n++) {
        buf = ecvt(PI / n, 10, &dec, &sign);
        sub(buf, dec, sign);
    }

    sub(char *buf, int dec, int sign)
    {
        int i;
        int n;

        (sign == 0) ? putchar('+') : putchar('-');
        if (dec > 0)
            n = dec;
        else
            n = 1;
        for (i = 0; buf[i] != '\0'; i++) {
            if (i == n)
                putchar('.');
            putchar(buf[i]);
        }
        if (dec <= 0)
            printf("e%d", dec - 1);
        putchar('\n');
    }
}
```


eof

レベル 2

書 式 `#include <io.h>`

関数宣言

```
int eof(handle) ;
int handle; /* 調べたいファイルのファイルハンドル */
```

機 能 handleで指定されるファイルが、エンドオブファイル (EOF) に達しているかどうかを調べます。

戻 り 値 現在の状態がエンドオブファイル (EOF) 状態である場合は1を返し、そうでなければ0を返します。

エラーが起きたときは、-1を返し errno に EBADF がセットされます。

EBADF はファイルハンドルが適切でないことを示します。

参 照 関 数 clearerr, feof, ferror

プログラム例

```
#include      <stdio.h>
#include      <stdlib.h>
#include      <fcntl.h>
#include      <stat.h>
#include      <io.h>

#define SIZE  16

main(int argc, char *argv[])
{
    char    buf[SIZE];
    char    *file;
    int     fn;
    int     n;

    file = (argc > 1) ? argv[1] : "file";

    if ((fn = open(file, O_RDONLY|O_BINARY)) == -1) {
        printf("can't open %s\n", file);
        exit(1);
    }
    while (!eof(fn)) {
        n = read(fn, buf, SIZE);
        printf("%d bytes readed.\n", n);
    }
    close(fn);
    exit(0);
}
```

execl, execl, execlp, execlpe, execv, execve, execvp, execvpe

レベル 2

書 式 #include <process.h>

```
int execl(pathname, arg0, arg1...argn, NULL) ;
int execl(pathname, arg0, arg1...argn, NULL, envp) ;
int execlp(pathname, arg0, arg1...argn, NULL) ;
int execlpe(pathname, arg0, arg1...argn, NULL, envp) ;
int execv(pathname, argv) ;
int execve(pathname, argv, envp) ;
int execvp(pathname, argv, envp) ;
int execvp(pathname, argv) ;
char *pathname; /* 実行ファイルのパス名へのポインタ */
char *arg0, *arg1, ..., *argn; /* 引数へのポインタリスト */
char *argv []; /* 引数へのポインタ配列 */
char *envp []; /* 環境へのポインタ配列 */
```

機 能

新しい子プロセスを起動して制御を移します。

メモリ内では新しい子プロセスは、呼び出したプロセスの次にロードされます。

このため、ロードができるだけのメモリ領域が必要です。

pathname は子プロセスのあるファイル名のパス名を指定します。

パス名に拡張子があれば、その拡張子で探しますが、指定がなければ、“X”ファイルを探します。

パス名がピリオドで終わっている場合は、拡張子のないファイルを探します。

引数はそのまま、新しいプロセスにわたされます。

パス名と引数あわせて 255 バイトまでで指定します。

文字列末のヌル文字もこのバイト数に含まれますので注意してください。

引数として、最小限、arg0 か argv [0] はわたさなくてはなりません。

execl, execl, execlp および execlpe (関数に“l”のつくもの) は、引数をポインタのリストでわたします。

arg0 は使用されません。

execv, execvp, execve および execvpe (関数に“v”のつくもの) は、引数をポインタの配列でわたします。

argv [0] は使用されません。

execl, execlpe, execve および execvpe (関数“e”のつくもの) は envp 引数で環境設定のリストをわたし、環境を指定します。

envp は文字列を指すポインタの配列で、各文字列は、

NAME = value

とします。

ここで、NAME は環境変数名で、value はその値です。

envp が NULL の場合、親プロセスの環境を引き継ぎます。

execvp、execlpe、execvp および execvpe (関数に“p”のつくもの) は、path 環境変数でディレクトリの指定ができます。

戻り値 正常終了した場合は正の数 (0 も) を返します。

これは起動したプロセスの終了コードです。

エラーが発生した場合は -1 を返します。

その場合 errno に下記の値がセットされます。

errno の値	内 容
E2BIG	引数リストが 256 バイトを超えている
EMFILE	オープンファイルの数が多すぎる
ENOENT	ファイルまたはパス名が見つからない
ENOEXEC	指定のファイルが実行できない
ENOMEM	子プロセスを実行するためのメモリが不足している

参 照 関 数 abort, exit, _exit, spawnl, spawnle, spawnlp, spawnv, spawnve, spawnvp, system

プログラム例

```

#include      <stdio.h>
#include      <stdlib.h>
#include      <process.h>

main(int argc, char *argv[])
{
    char      *args[4];
    char      *envs[3];
    char      *pathname;

    pathname = argv[1];
    args[0] = argv[0];
    args[1] = argv[2];
    args[2] = argv[3];
    args[3] = NULL;
    envs[0] = "path=A:%$bin$";
    envs[1] = "include=$$";
    envs[2] = NULL;

    if (execl(pathname, args[0], args[1], args[2], NULL) == -1)
        printf("失敗しました。%n");
    if (execv(pathname, args) == -1)
        printf("失敗しました。%n");
    if (execvp(pathname, args, envs) == -1)
        printf("失敗しました。%n");
    if (execvp(pathname, args) == -1)
        printf("失敗しました。%n");
}

```

except

レベル 2

書 式 `#include <math.h>`

```
double except(type, name, arg1, arg2, retval) ;
int type; /* 数値演算エラーのタイプ */
char *name; /* 例外条件が発生した関数名へのポインタ */
double arg1; /* 上記関数呼び出し時の引数 */
double arg2; /* 上記関数呼び出し時の引数 */
double retval; /* デフォルトの戻り値 */
```

機 能

標準数学ライブラリ関数は、例外条件が発生したときには `matherr` 関数を呼び出すように作られていますが、`matherr` 関数の呼び出しを簡略化するための関数として、`except` が用意されています。

`except` は、パラメータをセットして `matherr` 関数を呼び出し、`matherr` 関数の返す値に従って適当な値を返します。

`type` には、数値演算エラーのタイプを指定します。

詳細については、`matherr` 関数を参照してください。

`name` には、例外条件が発生した関数名を格納する文字列（ヌル文字で終了）へのポインタを指定します。

`arg1`、`arg2` は数値演算関数に指定された引数を保持しますが、引数が1つであれば、`arg1` に格納され、`arg2` は使用されません。

`retval` はエラーが発生した数値演算関数のデフォルトの戻り値です。

`type`、`name`、`arg1`、`arg2`、`retval` は構造体 `exception` の各メンバに格納され、その構造体へのポインタが `matherr` 関数に渡されます。

そして、`matherr` 関数からの戻り値が、例外回復処理を行ったことを示す値だったならば、構造体 `exception` のメンバ `retval` から値を取り出してそれを返します。

例外回復処理を行わなかったことを示す値だったならば、デフォルトの戻り値である `retval` を返します。

戻 り 値 `matherr` を呼び出して例外処理を行い、その結果を返します。

参 照 関 数 `matherr`

exit, _exit

レベル 2

書 式 #include <stdlib.h>

```
void exit(status) ;
void _exit(status) ;
int status; /*実行終了時のステータス */
```

機 能 実行を終了させます。

exit は、プログラムの終了前に atexit 関数で登録されていた関数を、登録した順と逆の順に呼び出します。

そしてすべてのバッファをフラッシュし、現在オープンしているすべてのファイルをクローズします。

_exit は何もせず、実行の終了のみを行います。

通常は、statusには、正常に終了する場合は定数として宣言されているEXIT_SUCCESSを、そうでない場合には定数として宣言されているEXIT_FAILUREをセットします。

戻 り 値 戻り値はありません。

参 照 関 数 abort, atexit

プログラム例

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

main(int argc, char *argv[])
{
    char    *file;
    FILE    *fp;

    file = (argc > 1) ? argv[1] : "file";
    if ((fp = fopen(file, "r")) == NULL) {
        printf("can't open %s\n", file);
        exit(EXIT_FAILURE);
    }
    (void)fclose(fp);
    printf("normal end\n");
    exit(EXIT_SUCCESS);
}
```

exp

レベル 2

書 式 #include <math.h>

```
double exp(x) ;  
double x; /* 浮動小数点値 */
```

機 能 自然対数の底 e の x 乗を返します。

戻 り 値 e^x を返します。オーバーフローが発生した場合、ERANGE を errno にセットします。
またエラー処理は、matherr 関数を使って行うことができます。

参 照 関 数 log

プログラム例

```
#include <stdio.h>  
#include <math.h>  
  
main()  
{  
    double x;  
    int i;  
  
    for (i = -32; i <= 32; i++) {  
        x = (double)i / 10.0;  
        printf("exp(%.4f) = %g\n", x, exp(x));  
    }  
}
```


fabs

レベル 2

書 式 #include <math.h>

```
double fabs(x) ;
double x; /* 浮動小数点値 */
```

機 能 浮動小数点値 x の絶対値を計算します。

戻 り 値 引数 x の絶対値を返します。

参 照 関 数 abs, cabs, labs

プログラム例

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

main()
{
    double x;
    int i;

    for (i = -32; i <= 32; i++) {
        x = (double)i / 10.0;
        printf("fabs(%+4.1f) = %g\n", x, fabs(x));
    }
}
```

fclose, fcloseall

レベル 2

書 式 #include <stdio.h>

```
int fclose(stream) ;  
FILE *stream; /* FILE 構造体へのポインタ */  
int fcloseall( ) ;
```

機 能

fcloseall は stdin、stdout、stderr、stdaux、stdprn 以外のオープン中のストリームをクローズします。

fclose は指定したストリームだけをクローズします。

ストリームに関連するすべてのバッファはフラッシュされ、またシステムが割りつけたバッファも解放されます。

ただし、setbuf を使って割りつけたバッファは解放されません。

stdin、stdout に対して fclose を行うと、入出力のリダイレクトが無効になりますので注意してください。

戻 り 値

fclose ではストリームを正常にクローズした場合は 0 を返し、fcloseall ではクローズしたストリームの数を返します。

正常にクローズできなかった場合は EOF を返します。

参 照 関 数

close, fdopen, fflush, fopen, freopen

プログラム例

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

main(int argc, char *argv[])
{
    char    *file;
    FILE    *fp;
    int      n;

    file = (argc > 1) ? argv[1] : "file";

    if ((fp = fopen(file, "r")) == NULL) {
        printf("%sを開けません。¥n", file);
        exit(1);
    }
    printf("%sを開きました。¥n", file);

    if (fclose(fp) != EOF) {
        printf("%sをcloseできません。¥n", file);
        exit(1);
    }
    printf("%sをcloseしました。¥n", file);

    n = fcloseall();
    printf("%d個のファイルをcloseしました。¥n", n);
    exit(0);
}
```

書 式 #include <stdlib.h>

```
char *fcvt(value, ndec, decptr, signptr) ;
double value; /* 変換すべき浮動小数点値 */
int ndec; /* 小数点以下の有効桁数 */
int *decptr; /* 小数点位置を返すポインタ */
int *signptr; /* 符号を返すポインタ */
```

機 能

浮動小数点値 value を文字列に変換し、末尾にヌル文字‘\0’をつけ加えます。

ndec は小数点以下の桁数で、value の小数点以下の桁数が ndec よりも大きい場合は四捨五入されます。小さいときは 0 をつけ加えます。

数字以外は変換されません。

decptr は左端から小数点の位置を示す整理値です。

0 または負ならば、小数点位置は左端となります。

signptr は value の符号で、0 の場合は正、それ以外の場合は負を示します。

文字列に変換できなかった時は、長さが 0 の文字列にします。

fcvt は、内部の静的な領域へのポインタを返しますので、返されたポインタが指す領域を書き変えてはいけません。

また、前回の呼び出して返された文字列は失われます。

なお、print などでも %f、%F の書式指定を使用すると fcvt が呼び出されるので、それ以前に fcvt で返された文字列は失われます。

戻 り 値

変換された文字列へのポインタを返します。

参 照 関 数

atof, ecvt, gcvt

プログラム例

```

#include      <stdio.h>
#include      <stdlib.h>

#define PI    3.1415926535897932385

main()
{
    int    dec;
    int    sign;
    char    *buf;
    int    n;

    for (n = 1; n <= 100; n++) {
        buf = fcvt(PI / n, 10, &dec, &sign);
        sub(buf, dec, sign);
    }
}

sub(char *buf, int dec, int sign)
{
    int    i;
    int    n;

    (sign == 0) ? putchar('+') : putchar('-');
    if (dec > 0)
        n = dec;
    else
        n = 1;
    for (i = 0; buf[i] != '\0'; i++) {
        if (i == n)
            putchar('.');
        putchar(buf[i]);
    }
    if (dec <= 0)
        printf("e%d", dec - 1);
    putchar('\n');
}

```

fdelete

レベル 3

書 式 #include <basic.h>

```
int fdelete(f_name) ;  
char *f_name; /* ファイル名格納領域へのポインタ */
```

機 能 f_name が指すファイルを削除します。

戻 り 値 成功すると 0 を、失敗すると -1 を返します。

fdopen

レベル 2

書 式 #include <stdio.h>

```
FILE *fdopen(handle, type);
int handle; /* 該当ファイルのファイルハンドル */
char *type; /* アクセスモード格納領域へのポインタ */
```

機 能 openなどでオープンした、handleで指定されたファイルをストリームに対応させます。
typeはストリームのアクセスモードを指定する文字列で、詳しくはfopen関数を参照してください。

戻 り 値 ストリームのポインタを返します。
エラーならばNULLを返します。

参 照 関 数 dup, dup2, fclose, fcloseall, fopen, freopen, open

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <stat.h>
#include <io.h>

main(int argc, char *argv[])
{
    char *file;
    FILE *fp;
    int fn;

    file = (argc > 1) ? argv[1] : "file";

    if ((fn = open(file, O_RDONLY|O_BINARY)) == -1) {
        printf("can't open %s\n", file);
        exit(1);
    }
    if ((fp = fdopen(fn, "r")) == NULL) {
        printf("%sをopenできません。%n", file);
        exit(1);
    }
    (void)fclose(fp);
    printf("normal end.%n");
    exit(0);
}
```

feof

レベル 2

書 式 #include <stdio.h>

```
int feof(stream);  
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能 指定されたストリームがエンドオブファイル (EOF) になったかどうかを調べます。

一度エンドオブファイルになると、ストリームがクローズするか rewind 関数を実行するまで、エンドオブファイル状態のままになります。

戻 り 値 現在の位置がエンドオブファイルでない場合は 0 を返します。
エンドオブファイルの場合は、0 以外の値を返します。

参 照 関 数 clearerr, eof, ferror

注: feof はマクロ定義と関数の両方があります。

マクロ定義で使いたい場合は、stdio.h をインクルードする前に次の 1 行を追加してください。

```
#define MACRO
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
  
main(int argc, char *argv[])  
{  
    char *file;  
    FILE *fp;  
    int fd;  
  
    file = (argc > 1) ? argv[1] : "file";  
  
    if ((fd = open(file, O_RDONLY | BINARY)) == -1) {  
        printf("can't open %s\n", file);  
        exit(1);  
    }  
  
    if ((fp = fdopen(fd, "r")) == NULL) {  
        printf("can't open %s\n", file);  
        exit(1);  
    }  
  
    void fclose(fp);  
    printf("normal end.\n");  
    exit(0);  
}
```


ferror

レベル 2

書 式 #include <stdio.h>

```
int ferror(stream) ;
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能 指定したストリームの入出力時にエラーが発生しているかどうかを判定します。エラーが発生した場合は、ストリームがクローズされるか、または rewind か clearerr 関数が呼ばれるまで、エラー状態のままになっています。

戻 り 値 エラーがない場合は 0 を、エラーがある場合は 0 以外の値を返します。

参 照 関 数 clearerr, eof, feof, fopen

プログラム例

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

main(int argc, char *argv[])
{
    char    *file;
    FILE    *fp;
    int     c;

    file = (argc > 1) ? argv[1] : "file";
    if ((fp = fopen(file, "r")) == NULL) {
        perror(file);
        exit(1);
    }
    while ((c = getc(fp)) != EOF) {
        if (ferror(fp)) {
            perror(file);
            clearerr(fp);
            break;
        }
    }
    (void)fclose(fp);
    exit(0);
}
```

注：ferror はマクロ定義と関数版があります。

マクロ定義で使いたい場合は、stdio.h をインクルードする前に次の 1 行を追加してください。

```
#define MACRO
```

fflush

レベル 2

書 式 `#include <stdio.h>`

```
int fflush(stream) ;  
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能 出力ストリームと関連するバッファ内のデータが、ファイルに書かれていないならば、それをファイルに書き込みます。

これにより、バッファが空になります。

書き込み後もストリームはオープンしたままです。

また、ファイルポインタも変化しません。

入力ストリームと関連するバッファ内にデータがあれば、それは捨てられバッファが空になります。

データを捨てるだけでストリームはオープンしたままです。

また、ファイルポインタも変化しません。

なお、stdin は、通常ではバッファを使っていませんが、Human68k 内部でバッファを持っていますので、fflush ではこれをフラッシュできません。

ファイルポインタとは、ファイル中のどの位置で読み書きが行われるかを保持している指標のことです。

バッファを使っていないストリームに対して、この関数を実行しても影響はありません。

stream に NULL を指定すると、すべての入出力ストリームに対して処理が行われます。

なお、クローズやフラッシュを行わずに exit 以外でプログラムを終了すると、最後にストリームに出力されたデータがバッファに残ったまま捨てられてしまいますので、必ず fflush してください。

戻 り 値 バッファ内のデータが正常に書き込めたときには 0 を返します。

指定のストリームがバッファを使っていないか、または読み出し専用でオープンしている場合にも 0 を返します。

エラーが起きたときは EOF を返します。

stream に NULL を指定したときは、常に 0 を返します。

参 照 関 数 fclose, fflush, setbuf

注：XC では fflush で NULL が指定できますが、他の C コンパイラではできない可能性があるので注意してください。

プログラム例

```

#include      <stdio.h>
#include      <stdlib.h>

main(int argc, char *argv[])
{
    char      *file;
    FILE      *fp;
    int        c;

    file = (argc > 1) ? argv[1] : "outfile";
    if ((fp = fopen(file, "w")) == NULL) {
        perror(file);
        exit(1);
    }
    while ((c = getchar()) != EOF) {
        putc(c, fp);
        if (ferror(fp)) {
            perror(file);
            clearerr(fp);
            break;
        }
    }
    if (fflush(fp) == EOF) {
        perror(file);
        clearerr(fp);
    }
    (void)fclose(fp);
    exit(0);
}

```

注：バッファがいっぱいになったとき、ストリームがクローズされたとき、ストリームをクローズしないでプログラムが exit によって終了したときには、自動的にフラッシュが行われます。

ただし、プログラム終了時に自動的にフラッシュしようとして発生したエラーは無視されてしまいますので、この場合には、書き込もうとしたデータが失われてしまう可能性があります。

fgetc, fgetchar

レベル 2

書 式 #include <stdio.h>

```
int fgetc(stream) ;  
FILE *stream; /* FILE 構造体へのポインタ */  
  
int fgetchar ( ) ;
```

機 能 fgetc は stream で指定するストリームから 1 文字読み込みます。
fgetchar は fgetc (stdin) と同じです。

戻 り 値 読み込んだ文字を返します。
エラー、またはエンドオブファイルのときは EOF を返します。
EOF が返された時にエラーかエンドオブファイルどちらが発生したかを判定するためには、feof または ferror 関数を使用してください。

参 照 関 数 fputc, fputchar, getc, getchar

プログラム例

```
#include <stdio.h>  
  
main(int argc, char *argv[])  
{  
    FILE *fp;  
    int c;  
  
    if (argc == 1) {  
        while ((c = fgetchar()) != EOF)  
            putchar(c);  
    }  
    else {  
        if ((fp = fopen(argv[1], "r")) == NULL) {  
            perror(argv[1]);  
            exit(1);  
        }  
        while ((c = fgetc(fp)) != EOF)  
            putchar(c);  
        (void)fclose(fp);  
    }  
    exit(0);  
}
```

fgetpos

レベル 2

書 式 #include <stdio.h>

```
int fgetpos(stream, pos);
FILE *stream; /* FILE 構造体へのポインタ */
fpos_t *pos; /* 求めたアクセス位置を格納するポインタ */
```

機 能 stream が指すストリームの現在のアクセス位置を求め、pos が指すポインタへ格納します。

異なる処理系の間で移植などを行う場合、処理系の違いなどにより、ストリームのアクセス位置を表す情報に違いがあるため、fseek や ftell 関数では十分なアクセスが行えない場合があります。

fgetpos は、そのような問題を解消し、移植しやすいプログラムを記述する為に用意されています。

XC では fpos_t 型は、stddef.h で以下のように型宣言されています。

```
typedef long fpos_t;
```

戻 り 値 成功すると 0 を、失敗すると 0 以外を返します。

参 照 関 数 fsetpos, ftell, fseek

```
#include <stdio.h>
#define SIZE 1024

main(int argc, char *argv[])
{
    char buf[SIZE];
    FILE *fp;
    fpos_t fpos;

    if (argc < 2) {
        fprintf(stderr, "Usage: %s filename\n", argv[0]);
        exit(1);
    }
    if ((fp = fopen(argv[1], "r")) == NULL) {
        perror("fopen");
        exit(1);
    }
    while (!feof(fp)) {
        if (fread(buf, 1, SIZE, fp) != SIZE)
            break;
        printf("%s", buf);
        fpos_t fpos;
        fgetpos(fp, &fpos);
        printf("fpos: %ld\n", fpos);
    }
    fclose(fp);
    exit(0);
}
```

fgets

レベル 2

書 式 #include <stdio.h>

```
char *fgets(string, n, stream) ;  
char *string; /* 読み込んだ文字列を格納する領域へのポインタ */  
int n; /* 読み込む文字数 */  
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能 streamで指定するストリームから改行文字‘ $\backslash n$ ’が現われるか、ストリームがエンドオブファイルになるか、または読み込んだ文字数が $n-1$ になるまで文字列を読み込みます。

読み込んだ文字列の末尾にはヌル文字‘ $\backslash 0$ ’が付加され、stringが指す領域に格納されます。

nが1のときは、読み込んだ文字列は空文字列になります。

戻 り 値 読み込んだ文字列へのポインタを返します。

エラーが発生した場合またはエンドオブファイルを読み込んだ場合はNULLを返します。

どちらによるNULLかは、ferrorまたはfeof関数を使用して調べることができます。

参 照 関 数 fputc, gets, puts

プログラム例

```
#include <stdio.h>  
  
#define SIZE 1024  
  
main(int argc, char *argv[])  
{  
    char buf[SIZE];  
    char *file;  
    FILE *fp;  
  
    file = (argc > 1) ? argv[1] : "file";  
    if ((fp = fopen(file, "r")) == NULL) {  
        perror(file);  
        exit(1);  
    }  
    while (fgets(buf, SIZE, fp) != NULL)  
        fputc(buf, stdout);  
    (void)fclose(fp);  
    exit(0);  
}
```

filelength

レベル 2

書 式 #include <io.h>

```
long filelength(handle) ;
int handle; /* 調べたいファイルのファイルハンドル */
```

機 能 指定した handle のファイルの大きさを調べます。

戻 り 値 ファイルの大きさをバイト数で返します。
エラーが起きたときは-1を返し、errnoにEBADFをセットします。

参 照 関 数 chsize, fileno

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>

main(int argc, char *argv[])
{
    char *file;
    FILE *fp;
    int len;

    while (--argc) {
        file = *++argv;
        if ((fp = fopen(file, "r")) == NULL) {
            perror(file);
            exit(1);
        }
        len = filelength(fileno(fp));
        printf("name = %s, length = %d\n", file, len);
        (void)fclose(fp);
    }
}
```

fileno

レベル 2

書 式 #include <stdio.h>

```
int fileno(stream) ;  
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能 指定したストリームと関連しているファイルハンドルを調べます。

戻 り 値 ファイルハンドルを返します。
複数のファイルハンドルがストリームと関連している場合は、ストリームが最初にオープンしたときに割りつけたファイルハンドルを返します。
ストリームがオープン済みのファイルを持たない場合の戻り値は不定です。

参 照 関 数 fdopen, filelength, fopen, freopen

プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
#include <io.h>  
  
main(int argc, char *argv[])  
{  
    char *file;  
    FILE *fp;  
    int len;  
  
    while (--argc) {  
        file = *++argv;  
        if ((fp = fopen(file, "r")) == NULL) {  
            perror(file);  
            exit(1);  
        }  
        len = filelength(fileno(fp));  
        printf("name = %s, length = %d\n", file, len);  
        (void)fclose(fp);  
    }  
}
```

注: fileno はマクロ定義と関数版とがあります。

マクロ定義で使いたい場合は、stdio.h をインクルードする前に次の 1 行を追加してください。

```
#define MACRO
```


files

レベル 2

書 式 #include <io. h>

```
char *files(buff, atr, datetime, len, pathname, src_atr);
char *buff; /* 検索用バッファへのポインタ */
char *atr; /* 見つけたファイルの属性を格納する領域へのポインタ */
int *datetime; /* 見つけたファイルの作成日時を格納する領域へのポインタ */
size_t *len; /* 見つけたファイルのバイト数を格納する領域へのポインタ */
char *pathname; /* 検索するファイルのパス名へのポインタ */
char src_atr; /* 検索する属性 */
```

機 能

pathname と src_atr で指定したファイルを検索し、見つけたファイルの情報を atr, datetime, len で指定した領域に格納し、戻り値として見つけたファイルのファイル名へのポインタを返します。

buff は、ファイルを検索するときにこの関数が使用するバッファへのポインタです。

バッファの容量は 150 バイト必要です。

atr は、見つけたファイルの属性を格納する領域へのポインタです。

atr の指す領域のサイズは 1 バイト以上必要です。

* atr に格納される属性は、各ビットが 1 の場合に以下の意味を表しています。

ビット 0 : 読み込み専用

ビット 1 : 隠しファイル

ビット 2 : システムファイル

ビット 3 : ボリューム名

ビット 4 : ディレクトリ

ビット 5 : 通常ファイル

datetime は、見つけたファイルの作成日時を格納する領域へのポインタです。

datetime の指す領域のサイズは 4 バイト以上必要です。

また、* datetime に格納される作成日時の形式は以下の通りです。

ビット 31~25 : 1980 年からの相対年数 (0~99)

ビット 24~21 : 月 (1~12)

ビット 20~16 : 日 (1~31)

ビット 15~11 : 時 (0~23)

ビット 10~5 : 分 (0~59)

ビット 4~0 : 秒/2(0~29)

len は、見つけたファイルのバイト数を格納する領域へのポインタです。

len の指す領域のサイズは 4 バイト以上必要です。

pathname には、検索したいファイルのパス名へのポインタを指定します。

パス名にはワイルドカードが使用できるので、“A: ¥*.¥” という設定も可能です。

src_atr は検索したいファイルの属性を指定します。

src_atr の形式は以下の通りです。

0x01 : 読み込み専用

0x02 : 隠しファイル

0x04 : システムファイル

0x08 : ボリューム名

0x10 : ディレクトリ

0x20 : 通常ファイル

この src_atr で示される属性と、ファイルの属性との論理積 (AND) を取った値が 0 でなければ、検索の対象となります。

したがって、すべての属性を検索の対象にする場合は、0x3F を指定します。

また、読み込み専用の通常ファイルを検索の対象にしようとして、0x21 を指定しても、すべての通常ファイルと、すべての読み込み専用のボリューム名、ディレクトリ、通常ファイルが対象となってしまうことに注意してください。

files 関数を実行した後で、nfiles 関数をエラーが返されるまで実行することで、pathname と src_atr で指定したファイルすべてを検索することができます。

戻り値

正常終了ならば、見つけたファイルのファイル名へのポインタを返します。

なお、このファイル名は buff で指定した検索用バッファに作成されるため、同一の検索用バッファを指定して files 関数や nfiles 関数を呼び出すと以前の内容が失われることに注意してください。

エラーならば NULL を返します。

参照関数

chgfa, chgft, nfiles

プログラム例

```

#include      <stdio.h>
#include      <stdlib.h>
#include      <io.h>

#define A_RDO      0x01      /* 読み込み専用 */
#define A_HDN      0x02      /* 隠しファイル */
#define A_SYS      0x04      /* システムファイル */
#define A_VOL      0x08      /* Volumeラベル */
#define A_DIR      0x10      /* ディレクトリ */
#define A_FIL      0x20      /* 通常ファイル */

static char      *pattern = "%$usr%$include%$.h";
static char      buf[150];
static char      atr;
static long      dt;
static size_t    len;

main()
{
    char      *name;

    name = files(buf, &atr, &dt, &len, pattern, A_FIL);
    if (name != NULL)
        printf("%s\n", name);
    else
        exit(1);

    while ((name = nfiles(buf, &atr, &dt, &len)) != NULL)
        printf("%s\n", name);

    exit(0);
}

```

書 式 #include <graph.h>

関数ライブラリ

```
void fill(x1, y1, x2, y2, p) ;
int x1; /* 始点 X 座標 */
int y1; /* 始点 Y 座標 */
int x2; /* 終点 X 座標 */
int y2; /* 終点 Y 座標 */
int p; /* パレットコード */
```

機 能 グラフィック画面に塗りつぶした長方形を描画します。

x1 には始点 X 座標を指定します (−32768~32767)。

y1 には始点 Y 座標を指定します (−32768~32767)。

x2 には終点 X 座標を指定します (−32768~32767)。

y2 には終点 Y 座標を指定します (−32768~32767)。

p にはパレットコードを指定します (0~65535)。

画面モードにより最大値が異なります。

戻 り 値 戻り値はありません。

プログラム例

```
#include <stdlib.h>
#include <basic0.h>
#include <graph.h>

main()
{
    int i;
    int x1, x2, y1, y2;
    int pc;

    b_init();
    screen(2, 0, 1, 1);
    vpage(1);
    for (i = 0; i < 500; i++) {
        x1 = rand() % 768;
        y1 = rand() % 512;
        x2 = rand() % 768;
        y2 = rand() % 512;
        pc = rand() % 16;
        fill(x1, y1, x2, y2, pc);
    }
    b_exit(0);
}
```

fix

レベル 3

書 式 #include <basic.h>

```
double fix(f) ;
double f; /* 浮動小数点値 */
```

機 能 浮動小数点値 f の整数部を求めます。

戻 り 値 浮動小数点値の整数部を返します。

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

int main()
{
    double x;
    int i;
    for (i = -20; i <= 20; i++) {
        x = (double)i / 10.0;
        printf("floor(%f) = %d\n", x, floor(x));
    }
}
```

floor

レベル 2

書 式 `#include <math.h>`

`double floor(x) ;`

`double x ;`

機 能 x 以下で最大の整数を求めます。

戻 り 値 double としての整数を返します。

参 照 関 数 `ceil`, `modf`

プログラム例

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

main()
{
    double  x;
    int     i;

    for (i = -20; i <= 20; i++) {
        x = (double)i / 10.0;
        printf("floor(%+4.1f) = %g\n", x, floor(x));
    }
}
```

flushall

レベル 2

書 式 `#include <stdio.h>`

```
int flushall( );
```

機 能 オープンされているすべてのストリームに対して fflush を行います。
ストリームはクローズされません。
また、ファイルポインタも変化しません。
詳細については、fflush 関数を参照してください。

戻 り 値 オープンされているストリームの総数を返します。
エラーを表す戻り値は返しません。

参 照 関 数 fflush

プログラム例

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

main()
{
    int    n;

    n = flushall();
    printf("%d個のstreamがopenされています。%n", n);
    exit(0);
}
```

注：バッファがいっぱいになったときや、ストリームがクローズされたとき、ストリームをクローズしないでプログラムが exit によって終了したときには、自動的にフラッシュが行われます。

ただし、プログラム終了時に自動的にフラッシュしようとして発生したエラーは無視されてしまいますので、この場合には、書き込もうとしたデータが失われてしまう可能性があります。

fmod

レベル 2

書 式 `#include <math.h>`

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

機 能 x/y の剰余 (浮動小数点値) を計算します。
 $x = i \times y + f$ となる浮動小数点値 f を返します。
ここで、 i は整数、 f は x と同じ符号の値です。

戻 り 値 計算結果を返します。
 y が 0 か x/y がオーバーフローになる場合は `errno` に `EDOM` がセットされ、0
を返します。
またエラー処理は、`matherr` 関数を使って行うことができます。

参 照 関 数 `ceil`, `fabs`, `floor`

プログラム例

```
#include <stdio.h>  
#include <math.h>  
  
main()  
{  
    double x;  
    double y;  
    int i;  
    int j;  
  
    for (i = -32; i <= 32; i++) {  
        for (j = -32; j <= 32; j++) {  
            x = (double)i / 10.0;  
            y = (double)j / 10.0;  
            printf("fmod(%+4.1f, %+4.1f) = %g\n",  
                x, y, fmod(x, y));  
        }  
    }  
}
```


fmode

レベル 2

書 式 #include <stdio.h>

```
void fmode(fp, mode);
FILE * fp; /* FILE 構造体へのポインタ */
int mode; /* ファイル変換モード */
```

機 能 fp が指すストリームのファイル変換モード（テキストかバイナリ）を変更します。

mode は 0 か 1 を指定します。

0 のときは変換モード（テキスト）になり、1 のときは無変換モード（バイナリ）になります。

テキストモードの場合、入力において復帰改行‘\r\n’を改行‘\n’に、出力において改行‘\n’を復帰改行‘\r\n’に変換します。

ファイル変換モードを変更する場合は、fopen か fflush 関数を行った直後にしてください。

戻 り 値 戻り値はありません。

fopen

レベル 2

書 式 #include <stdio.h>

```
FILE *fopen(pathname, type) ;  
char *pathname; /* ファイルのパス名へのポインタ */  
char *type; /* アクセスモード格納領域へのポインタ */
```

機 能

pathname で指定したファイルをオープンします。
type はファイルアクセスモードを指定する文字列で、次の通りです。

type の値	内 容
“r”	ファイルを読み出す（ファイルは存在していること）
“w”	書き込みのためにファイルを新規作成する（同じファイルがあると前の内容は失われる）
“a”	ファイルの最後に追加書き込みをする（ファイルがないと新規作成）
“r+”	更新（読み込みと書き込み）する（ファイルは存在していること）
“w+”	更新（読み込みと書き込み）のためにファイルを新規作成する（同じファイルがあると前の内容は失われる）
“a+”	読み込みと追加書き込みする（ファイルがないと新規作成）

type の指定は、ファイルの作成時のアクセスモードと矛盾してはいけません。
“a”または“a+”でファイルをオープンした場合には、ファイルポインタはファイルの最後へ移動し、オープン直後の書き込み操作はファイルの最後から行われます。

“a”、“a+”以外でオープンした場合には、ファイルポインタはファイルの先頭に位置します。

“r+”、“w+”、“a+”が指定された場合には、読み込みと書き込みの両方が可能となります（これをファイルの更新のためにオープンするといいます）。

しかし、読み込みから書き込みへ、またはその逆に移る場合には、fflush、fseek または rewind 関数によるバッファの書き出し操作が必要です。

ファイルポインタは fseek 関数により移動することができます。

なお、ファイルポインタとは、ファイル中のどの位置で読み書きが行われるかを保持している指標のことです。

さらに改行の変換モードを指定するために下記の文字を使用します。

type の値	内 容
"t"	テキストモードでオープンする 入力時には、復帰改行‘ <code>\r\n</code> ’を改行‘ <code>\n</code> ’に、出力時には改行‘ <code>\n</code> ’を復帰改行‘ <code>\r\n</code> ’に変換する
"b"	バイナリモードでオープンする

"t"または"b"が type 文字列にない場合、変換モードはデフォルトのモードを指示するグローバル変数 `_fmode` によって定義されます。

戻り値

オープンしたストリームのポインタを返します。

エラーが起きたときは `NULL` を返します。

参照関数

`fclose`, `fcloseall`, `fdopen`, `ferror`, `fileno`, `freopen`, `open`, `setmode`

プログラム例

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

main(int argc, char *argv[])
{
    char    *file;
    FILE    *fp;
    int     n;

    file = (argc > 1) ? argv[1] : "file";

    if ((fp = fopen(file, "r")) == NULL) {
        printf("%sを開けません。\\n", file);
        exit(1);
    }
    printf("%sを開きました。\\n", file);

    (void)fclose(fp);
    printf("%sをcloseしました。\\n", file);

    exit(0);
}
```

fprintf

レベル 2

書式

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

```
int fprintf(stream, format_string [, argument... ] );  
FILE *stream; /* FILE 構造体へのポインタ */  
char *format_string; /* 書式文字列へのポインタ */
```

機能

format_string で指定した書式に従って出力を変換し stream で指定されたストリームに出力します。

format_string に続く引数は書式に従って変換され出力されます。

format_string の書式と機能は printf 関数と同じです。

詳細については、printf 関数を参照してください。

戻り値

出力した文字数を返します。

参照関数

cprintf, fscanf, printf, sprintf

プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
  
main()  
{  
    char *file = "outfile";  
    FILE *fp;  
  
    if ((fp = fopen(file, "w")) == NULL) {  
        perror(file);  
        fp = stderr;  
    }  
    fprintf(fp, "string->[%s]\n", "hello");  
    fprintf(fp, "decimal->[%d]\n", 123);  
    fprintf(fp, "floating->[%g]\n", 3.14);  
    (void)fclose(fp);  
    exit(0);  
}
```

fputc, fputchar

レベル 2

書 式 #include <stdio.h>

```
int fputc(c, stream) ;
int c; /* 出力する文字 */
FILE *stream; /* FILE 構造体へのポインタ */
```

```
int fputchar(c) ;
int c; /* 出力する文字 */
```

機 能 fputc は、stream で指定するストリームに 1 文字出力します。
fputchar は、fputc(c, stdout) と同じになります。

戻 り 値 出力した文字を返します。
エラーが発生した場合は、EOF を返します。

参 照 関 数 fgetc, fgetchar, putc, putchar

プログラム例

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

main()
{
    int c;
    while ((c = getchar()) != EOF)
        fputc(c, stdout);
    exit(0);
}
```

fputs

レベル 2

書 式 #include <stdio.h>

```
int fputs(string, stream) ;  
char *string; /* 出力文字列へのポインタ */  
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能 stream で指定されたストリームに文字列 string を出力します。
文字列の末尾のヌル文字'¥0'は出力されません。

戻 り 値 正常終了したときは0を返します。
エラーが発生した場合は、0 以外を返します。

参 照 関 数 fgets, gets, puts, fputs

プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
  
#define SIZE 1024  
  
main(int argc, char *argv[])  
{  
    char buf[SIZE];  
    char *file;  
    FILE *fp;  
  
    file = (argc > 1) ? argv[1] : "file";  
    if ((fp = fopen(file, "r")) == NULL) {  
        perror(file);  
        exit(1);  
    }  
    while (fgets(buf, SIZE, fp) != NULL)  
        fputs(buf, stdout);  
    (void)fclose(fp);  
    exit(0);  
}
```

fread

レベル 2

書 式 #include <stdio. h>

```
size_t fread(buffer, size, count, stream) ;  
void *buffer; /* 読み込んだデータを格納する領域を指すポインタ */  
size_t size; /* データ項目のサイズ */  
size_t count; /* 読み込みデータ項目数 */  
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能

stream で指定されたストリームから size バイトの長さのデータ項目を count の数だけ読み込み、buffer が指す領域に格納します。
ファイルポインタは、読んだバイト数だけ進みます。
ストリームがテキストモードでオープンされている場合は、復帰改行‘\r\n’は改行‘\n’に変換されます。
この変換は、ファイルポインタや戻り値に影響を与えません。

戻 り 値

実際に読み込んだデータ項目の数を返します。
エラーが発生した場合や、指定したデータの項目数まで達しないうちにエンドオブファイル (EOF) になった場合には、count より少ない数が返されます。

参 照 関 数

fwrite, read

プログラム例

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

#define SIZE   1024

main(int argc, char *argv[])
{
    char    buf[SIZE];
    char    *file;
    FILE    *fp;
    int     n;

    file = (argc > 1) ? argv[1] : "file";
    if ((fp = fopen(file, "rb")) == NULL) {
        perror(file);
        exit(1);
    }
    while ((n = fread(buf, sizeof(char), SIZE, fp)) > 0)
        fwrite(buf, sizeof(char), n, stdout);
    (void)fclose(fp);
    exit(0);
}
```


free

レベル 2

書 式 #include <stdlib.h>

```
int free(ptr) ;
void *ptr; /* 解放すべきメモリ領域を指すポインタ */
```

機 能 calloc, malloc, または realloc 関数によって割りつけられたメモリを解放します。
ptr は解放すべきメモリ領域を指します。

戻 り 値 正常終了のときは0を返します。
エラーが発生した場合は0以外の値を返します。

参 照 関 数 calloc, malloc, realloc

プログラム例

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

#define B_SIZE 1024

main()
{
    int i;
    char *p;

    for (i = 1; i < 1024; i++) {
        printf("%4d block ", i);
        if ((p = malloc(i * B_SIZE)) == NULL) {
            printf("確保できませんでした。¥n");
            break;
        }
        printf("確保しました。");
        free(p);
        printf("; 解放しました。¥n");
    }
    exit(0);
}
```

注：無効な ptr (calloc, malloc, realloc 関数によって割りつけられたのではない領域を指すポインタ) を指定すると、その後の割つけに影響を及ぼし、エラーを起こす場合があります。

rename

レベル 3

書 式 #include <basic.h>

```
int rename(org, target) ;  
char *org; /* 変更前ファイル名格納領域へのポインタ */  
char *target; /* 変更後ファイル名格納領域へのポインタ */
```

機 能 org が指すファイルを、target が指すファイル名に変更します。

戻 り 値 成功すると 0 を、失敗すると -1 を返します。

```
#include <stdio.h>  
#include <stdlib.h>  
#define B_SIZE 1024  
  
main()  
{  
    int i;  
    char *p;  
  
    for (i = 1; i < 1024; i++)  
    {  
        printf("malloc block %d", i);  
        if (p = malloc(B_SIZE))  
        {  
            printf("確保できました。 %d", i);  
            break;  
        }  
        printf("確保できませんでした。");  
        free(p);  
        printf("解放しました。 %d", i);  
    }  
    exit(0);  
}
```

freopen

レベル 2

書 式 #include <stdio.h>

```
FILE *freopen(pathname, type, stream);
char *pathname; /* 新しいファイルのパス名を指すポインタ */
char *type; /* アクセスモード格納領域へのポインタ */
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能

stream で指定されたストリームをクローズし、pathname で指定したファイルを入出力ストリームに新たに割りつけます。

つまり stdin、stdout、stderr、stdaux、stdprn といったあらかじめオープンされているファイルを再割りつけするために使用します。

新たに再割りつけされたファイルは type で指定した型でオープンされます。

type の値とその内容は fopen 関数と同じです。

なお、stdin、stdout に対して freopen を行うと入出力のリダイレクトが無効になりますので注意してください。

戻 り 値

新しくオープンしたストリームのポインタを返します。

エラーが発生した場合は、もとのストリームがクローズされ、NULL が返されます。

参 照 関 数

fwrite, fread, fopen

プログラム例

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

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

    if (argc > 1) {
        if (freopen(argv[1], "r", stdin) == NULL) {
            perror(argv[1]);
            exit(1);
        }
    }
    while ((c = fgetchar()) != EOF)
        putchar(c);
    exit(0);
}
```

frexp

レベル 2

書 式 #include <math.h>

```
double frexp(x, expptr);  
double x; /* 浮動小数点値 */  
int *expptr; /* 指数を格納する領域へのポインタ */
```

機 能 浮動小数点値 x を仮数 m ($x = 2^n \times m$) と指数 n に分けます。

戻 り 値 仮数 m を返します。

m の範囲は、 $0.5 \leq |m| < 1$ になります。

指数 n は `expptr` が示す位置に格納されます。

x が 0 の場合は、仮数、指数ともに 0 になります。

参 照 関 数 `ldexp`, `modf`

プログラム例

```
#include <stdio.h>  
#include <math.h>  
  
main()  
{  
    double x;  
    double y;  
    int exp;  
    int i;  
  
    for (i = -10; i <= 10; i++) {  
        x = (double)i / 10.0;  
        y = frexp(x, &exp);  
        printf("(%.4lf) = %g * 2^%d\n", x, y, exp);  
    }  
}
```

fscanf

レベル 2

書 式 #include <stdio.h>

```
int fscanf (stream, format_string [, argument. . . ] );
FILE *stream; /* FILE 構造体へのポインタ */
char *format_string; /* 書式文字列へのポインタ */
```

機 能

stream で指定されたストリームからデータを読み込み、format_string の書式に従って変換し、後に続く argument の指すフィールドに代入します。

format_string の内容は scanf 関数と同じです。

また、引数 argument は format_string の各変換指定に対応する型変数のポインタです。

戻 り 値

代入したフィールドの数を返します。

代入しなかったフィールドは含みません。

代入したフィールドがない場合は 0 を返します。

読み込みの前に入力 that 尽きてしまった場合は EOF を返します。

参 照 関 数

fprintf, scanf, sscanf

プログラム例

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

main(int argc, char *argv[])
{
    void sample(FILE *);
    FILE *fp;
    if (argc == 1)
        sample(stdin);
    else {
        if ((fp = fopen(argv[1], "r")) == NULL) {
            perror(argv[1]);
            exit(1);
        }
        sample(fp);
        (void)fclose(fp);
    }
    exit(0);
}

static void sample(FILE *fp)
{
    char str[256];
    char c;
    long l;
    float d;

    fscanf(fp, "%s\n", str);
    fscanf(fp, "%c\n", &c);
    fscanf(fp, "%ld\n", &l);
    fscanf(fp, "%g", &d);
    printf("%s %c %ld %g\n", str, c, l, d);
}
```

fseek

レベル 2

書 式 #include <stdio.h>

```
int fseek(stream, offset, prev);
FILE *stream; /* FILE 構造体へのポインタ */
long offset; /* 起点からのオフセット */
int prev; /* ファイルポインタの起点 */
```

機 能

stream で指定したストリームのファイルポインタを prev で指定した起点から offset バイト分移動します。

prev の値と内容を以下に示します。

prev の値	内 容
SEEK_SET	ファイルの先頭
SEEK_CUR	ファイルポインタの現在位置
SEEK_END	ファイルの最後

テキストモードのストリームに対して、以下のいずれかの条件に一致しない fseek を行ってははいけません。

- offset が 0
- offset が ftell 関数で返された値であり、なおかつ prev が SEEK_SET

バイナリモードのストリームの場合は、ファイルの先頭より前でなければ、どこへ移動してもかまいません。

このとき、ファイルの最後より後へ移動すると、ファイルの最後の位置から移動後の位置まで、ファイルに 0 を書き足します。

戻 り 値

ファイルポインタが正しく移動したときには 0 を返します。

エラーのときは 0 以外を返します。

ターミナルやプリンタなどの、シークできないデバイスがストリームに対応している場合には戻り値は不定となります。

参 照 関 数

ftell, lseek, rewind

プログラム例

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

#define LEN 100L

main(int argc, char *argv[])
{
    char *file;
    FILE *fp;
    int c;
    long i, j;

    file = (argc > 1) ? argv[1] : "file";
    if ((fp = fopen(file, "rb")) == NULL) {
        perror(file);
        exit(1);
    }
    (void)fseek(fp, 0L, SEEK_END);
    j = ftell(fp);
    for (i = j - LEN + 1; i <= j; i++) {
        if (fseek(fp, i, SEEK_SET) != 0)
            break;
        else {
            c = fgetc(fp);
            putchar(c);
        }
    }
    (void)fclose(fp);
    exit(0);
}
```


fsetpos

レベル 2

書 式 #include <stdio. h>

```
int fsetpos(stream, pos) ;
FILE *stream; /* FILE 構造体へのポインタ */
fpos_t *pos; /* 移動したいファイル位置を格納してあるポインタ */
```

機 能 stream が指すストリームの現在のアクセス位置を、pos が指すポインタに格納されている位置に移動します。

異なる処理系の間で移植などを行う場合、処理系の違いなどにより、ファイルのアクセス位置を表す情報に違いがあるため、fseek や ftell 関数では十分なアクセスが行えない場合があります。

fsetpos は、そのような問題を解消し、移植しやすいプログラムを記述する為に用意されています。

XC では fpos_t 型は、stddef. h で以下のように型宣言されています。

```
typedef long fpos_t;
```

pos が指すポインタに格納されているファイル位置は、同一のファイル stream に対して行われた fgetpos 関数によって格納された値ではなくてはなりません。

戻 り 値 成功すると 0 を、失敗すると 0 以外を返します。

参 照 関 数 fgetpos, ftell, fseek

fstat

レベル 2

書 式 #include <stat. h>

```
int fstat(handle, buffer) ;
int handle; /* 該当ファイルのファイルハンドル */
struct stat *buffer; /* stat 構造体を指すポインタ */
```

機 能 handle で指定されるオープン中のファイルに関する情報を buffer が指す構造体に格納します。

stat 構造体は stat. h で以下のように定義されています。

```
struct stat {
    dev_t      st_dev; /* ファイルならばドライブ番号、デバイスならハンドラ番号 */
    ino_t      st_ino; /* 常に 0 */
    short      st_mode; /* ファイルタイプとファイル属性
                        デバイスならば S_IFCHR のビットがセットされ、ファイルならば S_IFREG のビットがセットされる
                        アクセスモードによってユーザー（リード／ライト）のビットがセットされる */
    short      st_nlink; /* 常に 1 */
    short      st_uid; /* 常に 0 */
    short      st_gid; /* 常に 0 */
    dev_t      st_rdev; /* ファイルならばドライブ番号、デバイスならハンドラ番号 */
    off_t      st_size; /* ファイルの大きさ（バイト数） */
    time_t     st_atime; /* 最終変更日時。time 関数の返回值と同じ表現 */
    time_t     st_mtime; /* 最終変更日時。time 関数の返回值と同じ表現 */
    time_t     st_ctime; /* 最終変更日時。time 関数の返回值と同じ表現 */
};
```

戻 り 値 ステータス情報を正しく得た場合には 0 を返します。
エラーが発生した場合は -1 を返し、EBADF を errno にセットします。

参照関数 access, chmod, filelength, stat

プログラム例

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stat.h>
#include <io.h>

main(int argc, char *argv[])
{
    struct stat    inf;
    char    *file;
    int    fn;

    file = (argc > 1) ? argv[1] : "file";
    if ((fn = open(file, O_RDONLY)) == -1) {
        perror(file);
        exit(1);
    }

    (void)fstat(fn, &inf);
    printf("name = %s, size = %d\n", file, inf.st_size);

    close(fn);
    exit(0);
}
```

注：handle がデバイスのとき、stat 構造体の st_size と st_atime、st_mtime、st_ctime のメンバは意味をもちません。

ftell

レベル 2

書 式 #include <stdio.h>

```
long ftell(stream);  
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能 stream で指定したストリームの現在のファイルポインタを調べます。
ファイルポインタは、ファイルの先頭からのバイト数で示されます。

戻 り 値 現在のファイルポインタを返します。
オープン時の type が“a”、“at”以外のオープンでストリームをオープンした直後は 0 を返します。
エラーが発生したときは -1 を返します。
ターミナルやプリンタなどのシークできないデバイスがストリームに対応している場合には戻り値は不定となります。

参 照 関 数 fseek, lseek, tell

プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
  
main(int argc, char *argv[])  
{  
    char *file;  
    FILE *fp;  
    int c;  
    long i, j;  
  
    file = (argc > 1) ? argv[1] : "file";  
    if ((fp = fopen(file, "rb")) == NULL) {  
        perror(file);  
        exit(1);  
    }  
    (void)fseek(fp, 0L, SEEK_END);  
    printf("file end -> %ld\n", ftell(fp));  
    (void)fclose(fp);  
    exit(0);  
}
```

ftime

レベル 2

書 式 #include <timeb.h>

```
void ftime(timeptr) ;
struct timeb *timeptr; /* timeb 構造体を指すポインタ */
```

機 能 現在の時刻を timeptr が指す構造体 timeb に格納します。

構造体 timeb は timeb.h で定義され、time, millitim, timezone, dstflag の4つのメンバからなります。

time にはグリニッジ標準時 1970 年 1 月 1 日 00:00:00 から経過した秒数がセットされます。

millitim には、常に 0 がセットされます。

timezone には、グリニッジ標準時と地方時との差（西回り）が分単位でセットされます。

timezone の値は、グローバル変数 timezone の値からとります。

dstflag は、グローバル変数 daylight の値により、夏時間調整が地方時に有効な場合、0 以外の値となります。

戻 り 値 戻り値はありません。

参 照 関 数 asctime, ctime, gmtime, time, tzset

プログラム例

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

main()
{
    struct timeb tbuf;

    ftime(&tbuf);
    printf("time->%ld\n", tbuf.time);
    printf("timezone->%d\n", tbuf.timezone);
    printf("dstflag->%d\n", tbuf.dstflag);
}
```

ftpack

レベル 2

書 式 #include <io.h>

```
int ftpack (dtlist) ;  
char *dtlist ; /* 年月日時分秒の格納領域へのポインタ */
```

機 能 dtlist で指定された年月日時分秒の情報を、日時情報に変換します。
年月日時分秒の情報は、dtlist の指す領域に以下の通りに格納します。

* (dtlist + 0) : 1980 年からの相対年数(0~99)

* (dtlist + 1) : 月(1~12)

* (dtlist + 2) : 日(1~31)

* (dtlist + 3) : 時(0~23)

* (dtlist + 4) : 分(0~59)

* (dtlist + 5) : 秒(0~59)

したがって、年月日時分秒の情報を格納する領域は6バイト必要です。

戻 り 値 変換した日時情報を以下に示すフォーマットで返します。

ビット 31~25 : 1980 年からの相対年数(0~99)

ビット 24~21 : 月(1~12)

ビット 20~16 : 日(1~31)

ビット 15~11 : 時(0~23)

ビット 10~5 : 分(0~59)

ビット 4~0 : 秒/2(0~29)

ftpack によって変換した日時情報は、chgft で使用することができます。

参 照 関 数 ftunpk, chgft, getft

プログラム例

```

#include      <stdio.h>
#include      <time.h>
#include      <io.h>
#include      <fcntl.h>
#include      <stat.h>

static char   dtlist[6] = {1990 - 1980, 3, 14, 12, 30, 0};

main()
{
    long       datetime;
    int        fn;

    if ((fn = open("¥¥CONFIG.SYS", O_RDWR)) == -1)
        puts("オープンできません。");

    datetime = ftpack(dtlist);
    chgft(fn, datetime);
}

```

書 式 #include <io. h>

```
void ftuspk(datetime, dtlist) ;
int datetime ;    /* 日時情報 */
char *dtlist ;    /* 年月日時分秒の格納領域へのポインタ */
```

機 能

datetime で指定された日時情報を年月日時分秒に変換し、dtlist の指す領域に格納します。

datetime のフォーマットは以下の通りです。

ビット 31～25 : 1980 年からの相対年数(0～99)

ビット 24～21 : 月(1～12)

ビット 20～16 : 日(1～31)

ビット 15～11 : 時(0～23)

ビット 10～5 : 分(0～59)

ビット 4～0 : 秒/2(0～29)

また、dtlist の指す領域には以下の通りに年月日時分秒の情報が格納されます。

* (dtlist + 0) : 1980 年からの相対年数(0～99)

* (dtlist + 1) : 月(1～12)

* (dtlist + 2) : 日(1～31)

* (dtlist + 3) : 時(0～23)

* (dtlist + 4) : 分(0～59)

* (dtlist + 5) : 秒(0～59)

したがって、年月日時分秒の情報が格納される領域は 6 バイト必要です。

getft で求めた日時情報を、ftunpk によって年月日時分秒に変換することができます。

戻 り 値 戻り値はありません。

参 照 関 数 ftpack, chgft, getft

プログラム例

```

#include <stdio.h>
#include <time.h>
#include <io.h>
#include <fcntl.h>
#include <stat.h>

main()
{
    long    datetime;
    int      fn;
    char     dtlist[6];

    if ((fn = open("%%CONFIG.SYS", O_RDWR)) == -1)
        puts("オープンできません。");
    datetime = getft(fn);
    ftunpk(datetime, dtlist);

    printf("%d/", dtlist[0] + 1980);
    printf("%d/", dtlist[1]);
    printf("%d ", dtlist[2]);
    printf("%d:", dtlist[3]);
    printf("%d:", dtlist[4]);
    printf("%d\n", dtlist[5]);
}

```

fwrite

レベル 2

書 式 #include <stdio.h>

```
size_t fwrite(buffer, size, count, stream) ;
void *buffer; /* 書き込むデータ格納領域へのポインタ */
size_t size; /* データ項目のサイズ */
size_t count; /* 書き込みデータ項目数 */
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能

buffer に格納されている size バイトの長さのデータ項目を count の数だけ、stream で指定されたストリームに書き込みます。
 ファイルポインタは、書き込んだバイト数だけ進みます。
 ファイルがテキストモードでオープンされている場合は、改行‘\n’は復帰改行‘\r\n’に変換されます。
 この変換はファイルポインタや戻り値には影響を与えません。

戻 り 値

実際に書き込んだデータ項目の数を返します。
 エラーが発生した場合には、count より少ない数が返されます。

参 照 関 数

fread, write

プログラム例

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

#define SIZE 1024

main(int argc, char *argv[])
{
    char buf[SIZE];
    char *file;
    FILE *fp;
    int n;

    file = (argc > 1) ? argv[1] : "file";
    if ((fp = fopen(file, "rb")) == NULL) {
        perror(file);
        exit(1);
    }
    while ((n = fread(buf, sizeof(char), SIZE, fp)) > 0)
        fwrite(buf, sizeof(char), n, stdout);
    (void)fclose(fp);
    exit(0);
}
```

gcvt

レベル 2

書 式 #include <stdlib.h>

```
char *gcvt (value, ndig, buffer);
double value; /* 変換すべき浮動小数点値 */
int ndig; /* 桁数 */
char *buffer; /* 変換文字列の格納領域を指すポインタ */
```

機 能 浮動小数点値 value を ndig の桁数で printf の %f フォーマット、すなわち指数のない形式に変換し、それが不可能な場合には %e フォーマット、すなわち指数を含んだ形式の文字列に変換し、buffer の指す領域に格納します。
文字列の末尾にはヌル文字 '\0' が付加されます。

戻 り 値 変換された文字列へのポインタを返します。

参 照 関 数 atof, atoi, ecvt, fcvt

プログラム例

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

#define PI 3.1415926535897932385

main()
{
    char    buf[256];
    int     n;

    for (n = 1; n <= 100; n++) {
        (void)gcvt(PI / n, 10, buf);
        printf("PI / %d = %s\n", n, buf);
    }
}
```

書 式 #include <graph.h>

```
void get (x1, y1, x2, y2, buf, size) ;  
int x1; /* 始点 X 座標 */  
int y1; /* 始点 Y 座標 */  
int x2; /* 終点 X 座標 */  
int y2; /* 終点 Y 座標 */  
void *buf; /* 画像データを読み込む領域へのポインタ */  
int size; /* 画像のバイト数 */
```

機 能 グラフィック画面の長方形の画像データを、size バイト数分、buf の指す領域に読み込みます。

x1 には始点 X 座標を指定します。

y1 には始点 Y 座標を指定します。

x2 には終点 X 座標を指定します。

y2 には終点 Y 座標を指定します。

始点と終点は、window 関数で指定したクリッピング座標の中でなければなりません。

buf には、読み込む画像データを格納する領域へのポインタを指定します。

この領域は、size バイト分なければいけません。

size には、画像データのバイト数を指定します。

画面モードにより、以下のバイト数を指定します。

16 色モード : (総ドット数+1) ÷ 2

256 色モード : 総ドット数

65536 色モード : 総ドット数×2

戻 り 値 戻り値はありません。

参 照 関 数 put, window

プログラム例

```

#include      <stdio.h>
#include      <basic0.h>
#include      <graph.h>

main()
{
    static char    buf[14015 + 1];
    FILE    *fp;

    b_init();
    screen(1, 3, 1, 1);
    vpage(1);
    symbol(3, 1, "x68000", 2, 2, 2, 65535, 0);
    box(0, 0, 147, 49, 65535, 0xffff);
    get(1, 1, 146, 48, buf, sizeof(buf));

    fp = fopen("a:test.dat", "w");
    fwrite(buf, 1, 14016, fp);
    (void)fclose(fp);
    b_exit(0);
}

```

getc, getchar

レベル 2

書 式 #include <stdio.h>

```
int getc(stream) ;  
FILE *stream; /* FILE 構造体へのポインタ */  
int getchar( ) ;
```

機 能 getc は stream で指定するストリームから 1 文字読み込みます。
getchar は getc (stdin) と同じです。

戻 り 値 読み込んだ文字を返します。
エラーが発生した場合またはエンドオブファイル (EOF) を読み込んだ場合は、EOF を返します。
どちらによる EOF かは、ferror または feof 関数を使用して調べることができます。

参 照 関 数 fgetc, fgetchar, getch, getche, putc, putchar, ungetc

プログラム例

```

#include      <stdio.h>
#include      <stdlib.h>

main(int argc, char *argv[])
{
    FILE      *fp;
    int       c;

    if (argc == 1) {
        while ((c = getchar()) != EOF)
            putchar(c);
    }
    else {
        if ((fp = fopen(argv[1], "r")) == NULL) {
            perror(argv[1]);
            exit(1);
        }
        while ((c = getc(fp)) != EOF)
            putchar(c);
        (void)fclose(fp);
    }
    exit(0);
}

```

注：getc, getchar は fgetc, fgetchar 関数と同じですが、マクロ定義と関数の両方があります。マクロ定義で使いたい場合は、stdio. h をインクルードする前に次の1行を追加してください。

```
#define MACRO
```

getch

レベル 2

書 式 #include <conio.h>

int getch();

機 能 コンソールから1文字読み込みます。
入力エコーバックはありません。

戻 り 値 読み込んだ文字を返します。

プログラム例

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

main()
{
    char *msg = "スペースキーを押してください。";

    while (cputs(msg), getch() != ' ')
        cputs("%r%n");
    cputs("%r%nOK%n");
    exit(0);
}
```


getche

レベル 2

書 式 #include <conio.h>

int getche();

機 能 コンソールから 1 文字読み込みます。入力エコーバックします。

戻 り 値 読み込んだ文字を返します。

参 照 関 数 cgets, getch, getchar

プログラム例

```
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>

main()
{
    char *msg = "0~9の番号を選んでください。";
    int c;

    while (cputs(msg), (c = getche()), !isdigit(c))
        cputs("%r%rn");
    cprintf("%r%rnselected -> [%c]%r%rn", c);
    exit(0);
}
```

getcwd

レベル 2

書 式 #include <direct. h>

```
char *getcwd(pathbuf, n) ;  
char *pathbuf; /* パス名を格納する領域を指すポインタ */  
int n; /* パス名の最大長 */
```

機 能 カレントディレクトリのフルパス名を pathbuf が指す領域に格納します。
パス名の最大長は n バイトで、パス名（最後のヌル文字を含めた）の長さが n の値を超えると、エラーになります。
pathbuf は NULL でもかまいません。
その場合、n の大きさのバッファを malloc 関数によって自動的に割りつけ、そこにパス名を格納します。
このバッファは後に、free 関数で getcwd の戻り値（割りつけたバッファのポインタ）を指定して解放することができます。

戻 り 値 ディレクトリのフルパス名へのポインタを返します。
エラーのときは NULL を返し、errno に次の値がセットされます。

errno の値	内 容
ENOMEN	n バイト分のメモリがない（NULL を引数として指定したとき）
ERANGE	パス名が n 文字より長くなっている

参 照 関 数 chdir, mkdir, rmdir

プログラム例

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

#define SIZE 1024

static char buf[SIZE];
```

```
main()
{
    if (getcwd(buf, SIZE) == NULL)
        perror("getcwd error");
    else
        printf("cwd is '%s'\n", buf);
}
```

読み出し

参照関係

関数呼び出し

getenv

レベル 2

書 式 #include <stdlib.h>

```
char *getenv(name) ;  
char *name; /* 環境変数名を指すポインタ */
```

機 能 環境変数リストから name で指定される環境変数名の内容を獲得します。

getenv で返された環境変数リストのエントリを直接変更してはいけません。

エントリを変更する場合には、putenv 関数を使用します。

また、環境変数リストに影響を与えずに返された文字列を加工したい場合は、strdup, strcpy などの関数を使って文字列のコピーを作成して、コピーに対して加工してください。

getenv, putenv 関数は環境変数リストをアクセスするためにグローバル変数 environ を使用します。

戻 り 値 環境変数の内容へのポインタを返します。
環境変数を発見できなかった場合は NULL を返します。

参 照 関 数 putenv

プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
  
#define NAME "path"  
  
main()  
{  
    char *p;  
  
    if ((p = getenv(NAME)) == NULL)  
        printf("%s is not found.\n", NAME);  
    else  
        printf("%s = %s\n", NAME, p);  
}
```

getfa

レベル 2

書 式 #include <io.h>

```
int getfa(pathname) ;
char *pathname; /* 属性を調べたいファイルのパス名へのポインタ */
```

機 能 pathnameで指定されたファイルの属性を調べます。

戻 り 値 正常終了ならば指定されたファイルの属性を返します。
属性のフォーマットは以下の通りです。

ビット0 : 読み込み専用

ビット1 : 隠しファイル

ビット2 : システムファイル

ビット3 : ボリューム名

ビット4 : ディレクトリ

ビット5 : 通常ファイル

なお、ビット3～5までのいずれのビットも付いていない場合は通常ファイルです。

エラーならば-1を返します。

参 照 関 数 chgfa

プログラム例

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

#define A_RDO      0x01      /* 読み込み専用 */
#define A_HDN      0x02      /* 隠しファイル */
#define A_SYS      0x04      /* システムファイル */
#define A_VOL      0x08      /* Volumeラベル */
#define A_DIR      0x10      /* ディレクトリ */
#define A_FIL      0x20      /* 通常ファイル */

main()
{
    int      r;

    if ((r = getfa("¥¥CONFIG.SYS")) == -1)
        puts("失敗しました。");
    else {
        if (r & A_RDO)
            puts("読み込み専用");
        if (r & A_HDN)
            puts("隠しファイル");
        if (r & A_SYS)
            puts("システムファイル");
        if (r & A_VOL)
            puts("Volumeラベル");
        if (r & A_DIR)
            puts("ディレクトリ");
        if (r & A_FIL)
            puts("通常ファイル");
    }
}
```

getft

レベル 2

書 式 #include <io.h>

```
int getft (file) ;
int file; /* 作成日時を調べたいファイルのファイルハンドル */
```

機 能 fileで指定されたファイルの作成日時を調べます。

戻 り 値 正常終了ならば、指定されたファイルの作成日時を返します。
作成日時のフォーマットは以下の通りです。

ビット31～25 : 1980年からの相対年数(0～99)

ビット24～21 : 月(1～12)

ビット20～16 : 日(1～31)

ビット15～11 : 時(0～23)

ビット10～5 : 分(0～59)

ビット4～0 : 秒/2(0～29)

エラーならば-1を返します。

なお、作成日時は負の数となる場合もあるので、エラーかどうかを単に戻り値の符号で判別できないことに注意してください。

この返り値は、ftunpkによって年月日時分秒に変換することができます。

参 照 関 数 chgft, ftunpk

プログラム例

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <stat.h>

main()
{
    int    date;
    int    fn;

    if ((fn = open("¥¥CONFIG.SYS", O_RDWR)) == -1)
        puts("オープンできません。");

    if ((date = getft(fn) == -1L)
        puts("失敗しました。");
    else
        puts(dtasc(date >> 16));
    close(fn);
}
```

getl

レベル 2

書 式 #include <stdio.h>

int getl (fp) ;

FILE *fp; /* ファイル構造体へのポインタ */

機 能 fp で指定されたストリームから long サイズ分読み込みます。

戻 り 値 読み込んだ long の値を返します。
エラーが発生した場合またはエンドオブファイル (EOF) を読み込んだ場合は、EOF を返します。
ただし、ストリームから読み込んだデータが EOF と一致している可能性がありますので、エラーとエンドオブファイルの判定は ferror と feof 関数を使用して判定してください。

参 照 関 数 putw, putl, getw

参 照 関 数

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

main()
{
    int fd;
    int i;

    if ((fd = open("COMP10.278", O_RDWR)) == -1)
        puts("オープンできません。");

    if ((date = read(fd, 1)) == -1)
        puts("失敗しました。");
    else
        puts(read(date) >> 16);

    close(fd);
}
```


getmem, rlsmem, getml, rlsml

レベル 2

書 式 #include <stdlib.h>

```
void *getmem(size) ; /* メモリ領域の確保 */
size_t size; /* 確保するメモリ領域のサイズ */

int rlsmem(p, size) ; /* メモリ領域の解放 */
char *p; /* 解放する領域へのポインタ */
size_t size; /* 解放するメモリ領域のサイズ */

void *getml(size) ; /* メモリ領域の確保 */
long size; /* 確保するメモリ領域のサイズ */

int rlsml(p, size) ; /* メモリ領域の解放 */
char *p; /* 解放する領域へのポインタ */
long size; /* 解放するメモリ領域のサイズ */
```

機 能 メモリ領域の確保、または解放を行います。

getmem は、size バイトの領域を確保し、その領域を指すポインタを返します。
メモリが不足して領域を確保できないときには NULL を返します。

rlsmem は、getmem で確保したメモリ領域を解放します。

getml, rlsml は、引数の型が long である他は、それぞれ getmem, rlsmem と同じです。

rlsmem で指定するサイズ size は、getmem で指定したものと等しくなければなりません。getmem 以外の関数で確保した領域を解放しようとした場合、rlsmem は -1 を返します。

戻 り 値 処理が正常に終了した場合、getmem, getml は確保したメモリ領域へのポインタを、rlsmem, rlsml は 0 を返します。

エラーが発生した場合には、getmem, getml は NULL を、rlsmem, rlsml は -1 を返します。

プログラム例

```

#include      <stdio.h>
#include      <stdlib.h>

#define B_SIZE 1024

main()
{
    int      n;
    char     *p;

    for (n = 1; n <= 1000; n++) {
        printf("%dKbyte: ", n);
        if ((p = getmem(n * B_SIZE)) != NULL) {
            printf("確保できました。");
            rlsmem(p, n * B_SIZE);
            printf(": 全メモリを解放しました¥n");
        }
        else {
            printf("確保できませんでした。¥n");
            break;
        }
    }
}

```

解 説

このプログラムは、メモリの確保と解放を繰り返す。getmemは、sizeバイトの領域を確保し、その領域のポインタを返す。もしもメモリが不足して領域を確保できなかった場合はNULLを返す。rlsmemは、getmemで確保したメモリ領域を解放する。getml, rlsmlは、引数の型がlongである場合、その型はgetmem, rlsmemと同じです。

rlsmemで指定するサイズは、getmemで指定するサイズと同じである。サイズが異なる場合、getmem以外の関数で確保した領域を解放しようとするとエラーが返ります。

関 数

関数が正常に終了した場合、getmem, getml, rlsmem, getml, rlsmlはメモリを解放し、その領域は再利用されます。エラーが返った場合は、getmem, getml, rlsmem, getml, rlsmlはメモリを解放し、その領域は再利用されません。

getpid

レベル 2

書 式 #include <process.h>

```
int getpid( ) ;
```

機 能 呼び出しプロセスを識別するプロセス ID を返します。

プロセス ID には、プログラム実行開始時にそのプロセスに固有な数値が与えられます。

戻 り 値 プロセス ID が返ります。

参 照 関 数 mktemp

プログラム例

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

```
main()
{
    char    name[256];

    sprintf(name, "%tmp%file%04d", getpid());
    printf("temporary file name = %s\n", name);
}
```

gets

レベル 2

書 式 #include <stdio.h>

```
char *gets(buffer);  
char *buffer; /* 読み込んだ文字を格納する領域へのポインタ */
```

機 能

標準入力から改行文字'\n'が現われるか、標準入力からエンドオブファイル (EOF) になるまで文字列を読み込みます。

読み込んだ文字列の末尾にはヌル文字'\0'が付加され、buffer が指す領域に格納されます。

戻 り 値

読み込んだ文字列へのポインタを返します。

エラーが発生した場合またはエンドオブファイル (EOF) を読み込んだ場合は、NULL を返します。

どちらによる NULL かは、ferror または feof 関数を使用して調べることができます。

参 照 関 数

fgets, fputs, puts

プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
  
#define SIZE 4096  
  
static char buf[SIZE];  
  
main()  
{  
    while (gets(buf) != NULL)  
        puts(buf);  
    if (ferror(stdin)) {  
        perror("read error");  
        abort();  
    }  
    exit(0);  
}
```

getw

レベル 2

書 式 #include <stdio.h>

```
int getw(fp);
```

FILE *fp; /* FILE 構造体へのポインタ */

機 能 fp で指定されたストリームから short サイズ分読み込みます。

戻 り 値 読み込んだ short の値を返します。
エラーが発生した場合またはエンドオブファイル (EOF) を読み込んだ場合は、EOF を返します。

どちらによる EOF かは ferror または feof 関数を使用して調べることができます。

参 照 関 数 putw, putl, getl

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>

main()
{
    short    s;
    FILE     *fp;

    fp = stdin;

    if ((s = getw(fp)) == EOF) {
        if (ferror(fp)) {
            perror("fail");
            clearerr(fp);
        }
        else if (feof(fp))
            printf("end of file\n");
        exit(1);
    }
    printf("number = %d\n", s);
    exit(0);
}
```

書 式 #include <time. h>

```
struct tm *gmtime(time) ;
time_t *time; /* 時刻データの格納領域を指すポインタ */
```

機 能 時刻 time をグリニッジ標準時に変換し、tm 構造体に格納します。
時刻 time はグリニッジ標準時 1970 年 1 月 1 日 0 時 0 分 0 秒から経過した秒数で、time を呼び出すことにより得られます。
tm 構造体は、time. h で以下のように宣言されています。

```
struct tm {
    int tm_sec;      /* 秒(0~59) */
    int tm_min;      /* 分(0~59) */
    int tm_hour;     /* 時間(0~23) */
    int tm_mday;     /* 日付(1~31) */
    int tm_mon;      /* 月(0~11、0 が 1 月) */
    int tm_year;     /* 年(現在の年から 1900 を引いた数) */
    int tm_wday;     /* 曜日(0~6、0 が 日曜日) */
    int tm_yday;     /* 1 年を通した日付 (0~365、0 は 1 月 1 日) */
    int tm_isdst;    /* 夏時間調整が無効の場合は 0、
                       有効の場合は 0 以外 */
};
```

gmtime は、内部の静的な領域へのポインタを返します。
したがって、gmtime が呼び出されるたびに同じ領域に上書きされ、前の内容は失われます。

戻 り 値 tm 構造体を指すポインタを返します。

参 照 関 数 asctime, ctime, localtime, time

プログラム例

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

main()
{
    struct tm    *l_tm;
    struct tm    *u_tm;
    time_t    ltime;

    time(&ltime);

    u_tm = gmtime(&ltime);
    printf("ただ今の UCT 時刻は %s\n", asctime(u_tm));

    l_tm = localtime(&ltime);
    printf("ただ今の 現地時刻は %s\n", asctime(l_tm));
}
```

hantozen

レベル 2

書 式 #include <jfctype.h>

```
int hantozen(c) ;  
int c; /* 変換する文字 (半角文字) */
```

機 能 半角文字 c を全角文字に変換します。
 c は $0x20 \leq c \leq 0 \times 7E$ 、または $0xA1 \leq c \leq 0xDF$ の範囲内になければなりません。
カナはカタカナに変換されます。
 c のビット 16 が 1 のときには、ひらがなに変換されます。

戻 り 値 シフト JIS 漢字コードでの全角文字を返します。
変換できなかった場合には c を返します。

参 照 関 数 zentohan

home

レベル 3

書 式 #include <graph.h>

```
void home(pa, x, y) ;
int pa; /* グラフィックページ */
int x; /* 左端 X 座標 */
int y; /* 上端 Y 座標 */
```

機 能 グラフィック画面の表示位置を設定します。
 pa にはグラフィック画面のページを指定します (0~3)。
 ただし、画面のモードにより、最大ページ数が異なります。
 x には左端 X 座標、y には上端 Y 座標を指定します。
 x、y もまた、画面のモードにより最大値が異なります (0~511、または0~1023)。

戻 り 値 戻り値はありません。

プログラム例

```
#include <basic0.h>
#include <graph.h>

main()
{
    b_init();

    screen(2, 0, 1, 1);
    apage(0);
    vpage(1);
    window(0, 0, 1023, 1023);
    circle(511, 511, 100, 15, 0, 360, 256);
    paint(511, 511, 9);

    home(0, 255, 255);

    b_exit(0);
}
```

hsv

レベル 3

書 式 #include <graph.h>

```
int hsv(hu, sa, va) ;  
int hu; /* 色相 */  
int sa; /* 濃さ */  
int va; /* 明るさ */
```

機 能 色相 hu, 濃さ sa, 明るさ va に応じた色コードを得ます。
なお、色コードは 0~65534 の偶数となります。
それぞれの指定可能範囲は以下の通りです。

hu : 0~192
sa : 0~31
va : 0~31

戻 り 値 色コードを返します。

プログラム例

```

#include <basic0.h>
#include <graph.h>

main()
{
    int i, j, k;
    int kyu(int, int, int);

    b_init();
    screen(1, 3, 1, 1);
    vpage(1);
    for (i = 0; i < 16; i++) {
        for (j = 0; j < 4; j++) {
            for (k = 0; k < 3; k++) {
                kyu(j * 128, k * 170,
                    i * 12 + j * 3 + k);
            }
        }
    }
    b_exit(0);
}

static int kyu(int x, int y, int h)
{
    int i, j, k;
    int c;

    fill(x, y, x + 127, y + 169, 0);
    for (i = 0; i < 32; i++) {
        j = 31 - i;
        for (k = 0; k < 2; k++) {
            c = hsv(h, 31, i) + k;
            circle(x + 64, y + 64,
                j * 2 + k, c, 0, 360, 256);
            paint(x + 64, y + 64, c);
        }
    }
}

```

hypot

レベル 2

書 式 `#include <math.h>`

```
double hypot(x, y);  
double x, y; /* 直角三角形の 2 辺 */
```

機 能 2 辺 x, y の長さから直角三角形の斜辺の長さを計算します。
計算式は $\sqrt{x^2 + y^2}$ と同じです。

戻 り 値 直角三角形の斜辺の長さを返します。
オーバーフローが発生した場合は、errno に ERANGE がセットされ返り値は HUGE_VAL になります。
またエラー処理は、matherr 関数を使って行うことができます。

参 照 関 数 cabs

プログラム例

```
#include <stdio.h>  
#include <math.h>  
  
main()  
{  
    double x;  
    double y;  
    int i;  
    int j;  
  
    for (i = 1; i <= 32; i++) {  
        for (j = 1; j <= 32; j++) {  
            x = (double)i;  
            y = (double)j;  
            printf("hypot(%.1f, %.1f) = %g\n",  
                x, y, hypot(x, y));  
        }  
    }  
}
```

img_color

レベル 3

書 式

#include <image.h>

```
void img_color(hu_cal) ;  
char hu_cal; /* 色調モード */
```

機 能

画像とり込みを行う際のとり込み画像の色調補正（グラフィックパレットの補正）を行います。

hu_cal には次の色調モードを指定します。

- 0 : 16 色モード
- 1 : 256 色モード
- 2 : 65536 色モード

戻 り 値

戻り値はありません。

img_home

レベル 3

書 式 #include <image. h>

```
void img_home(x, y, dir, cnt, wt) ;  
int x, y;  
char dir;  
int cnt, wt;
```

機 能

グラフィック画面の表示位置を移動します。

x には始点の X 座標 (0~511)、y には始点の Y 座標 (0~511) を指定します。

dir, cnt, wt はそれぞれ、移動方向 (1~9)、移動回数 (0~65535)、待ち時間 (0~65535) です。

戻 り 値

戻り値はありません。

img_ht

レベル 3

書 式 #include <image.h>

```
void img_ht(x1, y1, x2, y2, p, a);
int x1; /* 始点 X 座標 */
int y1; /* 始点 Y 座標 */
int x2; /* 終点 X 座標 */
int y2; /* 終点 Y 座標 */
char p; /* グラフィック画面の機能設定 */
char a; /* 半透明表示指定 */
```

機 能

グラフィック画面の指定範囲とテレビ画面との半透明表示の制御を行います。

x1 には始点 X 座標を指定します。

y1 には始点 Y 座標を指定します。

x2 には終点 X 座標を指定します。

y2 には終点 Y 座標を指定します。

p にはグラフィック画面の指定領域への半透明表示の設定有無を指定します。

0 : 半透明表示を設定しない

1 : 半透明表示を設定する

a には半透明表示の実行の有無を指定します。

0 : 半透明表示を実行しない

1 : 半透明表示を実行する

本関数実行前に、screen 関数または crt 関数により、スーパーインポーズの低解像度モード (512×512、または 256×256) にしておく必要があります。

また、x1、y1、x2、y2 は実画面サイズにより、512×512 のときは、0～511 まで、256×256 のときは、0～255 までを指定することができます。

戻 り 値

戻り値はありません。

img_load

レベル 3

書 式 #include <image.h>

```
int img_load(fname, x, y, b);  
char *fname; /* 入力ファイル名へのポインタ */  
int x; /* ロードするグラフィック画面の X 座標 */  
int y; /* ロードするグラフィック画面の Y 座標 */  
int b; /* ロード終了位置の設定 */
```

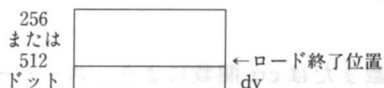
機 能 fname が指すファイルのデータを、グラフィック画面へ読み込みます。
ファイルの拡張子は、解像度により次のように決まっています。

拡張子	画面のモード	色
GS0	256×256	16 色
GM0	256×256	256 色
GL0	256×256	65536 色
GS3	512×512	16 色
GM3	512×512	256 色
GL3	512×512	65536 色

b: ロード終了位置

$$b = \frac{dy}{n}$$

dy…下図の通り Y 座標の下からのドット数



256 または 512 ドット

n…拡張子

GS0	……32	} 0 ≤ b ≤ 7
GS3	……16	
GM0	……16	
GM3	……8	
GL0	……8	
GL3	……	使用不可

戻 り 値 正常終了のときは 0 を、異常終了のときは 0 以外を返します。

参 照 関 数 img_save

img_pos

レベル 3

書 式

#include <image.h>

```
void img_pos(h_cal);
char h_cal; /* 画面水平位置補正データ */
```

機 能

画像とり込み時の画面水平位置の補正を行います。

h_cal は、0~255 までの範囲で指定します。

戻 り 値

戻り値はありません。

img_put

レベル 3

書 式 #include <image. h>

```
void img_put(x1, y1, x2, y2, na, len) ;  
int x1, y1, x2, y2, len ;  
void *na ;
```

機 能

グラフィックデータを表示します。

(x1, y1) で表される座標から (x2, y2) で表される座標までの間、8 ドットごとに移動しながら、na が指すパターンデータを表示します。

表示するパターンデータの大きさは 256×256 ドットです。

表示データは、131,072 バイト必要です。

戻 り 値

戻り値はありません。

img_save

レベル 3

書 式 #include <image. h>

```
int img_save(fname, x, y) ;
char *fname; /* 出力ファイル名へのポインタ */
int x; /* セーブするグラフィック画面の X 座標 */
int y; /* セーブするグラフィック画面の Y 座標 */
```

機 能 グラフィック画面のデータを、fname が指すファイルへ書き出します。
ファイルの拡張子は、解像度により次のように決まっています。

拡張子	画面のモード	色
GS0	256×256	16 色
GM0	256×256	256 色
GL0	256×256	65536 色
GS3	512×512	16 色
GM3	512×512	256 色
GL3	512×512	65536 色

戻 り 値 正常終了のときは 0 を、異常終了のときは 0 以外を返します。

参 照 関 数 img_load

img_scrn

レベル 3

書 式 #include <image. h>

void img_scrn(size, mode, disp) ;
char size, mode, disp ;

機 能

グラフィック画面のモードを設定します。
引数はそれぞれ次のように指定します。

size (表示画面サイズ)

0 : 256×256 ドット

1 : 512×512 ドット

2 : 768×512 ドット

mode (実画面および色モード)

0 : 1024×1024 ドット

16 色

1 : 512×512 ドット

16 色

2 : 512×512 ドット

256 色

3 : 512×512 ドット

65536 色

disp (ディスプレイ解像度)

0 : 標準解像度

1 : 高解像度

戻 り 値

戻り値はありません。

img_set

レベル 3

書 式 #include <image.h>

```
void img_set(img_ctl0[, img_ctl1, img_ctl2, img_ctl3, img_ctl4]) ;  
char img_ctl0; /* 制御符号 */  
int img_ctl1, img_ctl2, img_ctl3, img_ctl4; /* 制御符号 */
```

機 能 カラーイメージユニット(CZ-6VT1)の制御を行います。
img_ctl には、カラーイメージユニットのコントロールコードを指定します
(0x00~0xFF)。

戻 り 値 戻り値はありません。

img_still

レベル 3

書 式 `#include <image.h>`

```
void img_still(ch) ;  
char ch; /* モード */
```

機 能 画像とり込みの設定を行います。

ch には画像とり込みのモードを指定します。

0 : 実行 (静止画)

1 : 待機 (動画)

本関数の実行前に、screen 関数により低解像モード (512×512、または 256×256) にしておく必要があります。

また、画像とり込みを行うプログラムの終了時には、必ず `img_still(0)` を実行してください。

戻 り 値 戻り値はありません。

instr

レベル 3

書 式 #include <basic.h>

```
int instr(i, str1, str2) ;
int i; /* 比較を開始する位置 */
char *str1; /* 文字列 */
char *str2; /* 文字列 */
```

機 能 str1 の i 番目の文字から始めて、str2 と等しい部分を探します。
i に指定する値は、先頭を 1 として何番目かを数値で指定します。

戻 り 値 str1 に str2 と等しい部分のあった位置を返します。
返される値は、先頭を 1 とした数値を返します。
なければ 0 を返します。

isalkana, ispnkana, isalnmkana, isprkana, isgrkana

レベル 2

書 式 #include <jfctype.h>

```
int isalkana(c) ; /* 英文字またはカナ文字の判定 */
int ispnkana(c) ; /* 英句読点またはカナ句読点の判定 */
int isalnmkana(c) ; /* 英数字またはカナ文字の判定 */
int isprkana(c) ; /* 表示可能文字 (空白を含む) の判定 */
int isgrkana(c) ; /* 表示可能文字 (空白を除く) の判定 */
int c ; /* 判定する文字 */
```

機 能 これらの関数は c で与えられた文字の判定を行います。

isalkana は c が英文字またはカナ文字であるか否かを判定します。

ispnkana は c が英句読点またはカナ句読点であるか否かを判定します。

isalnmkana は c が英数字またはカナ文字であるか否かを判定します。

isprkana は c が表示可能文字 (空白を含む) であるか否かを判定します。

isgrkana は c が表示可能文字 (空白を除く) であるか否かを判定します。

戻 り 値 条件を満たせば 0 以外の値を、満たさなければ 0 を返します。

参 照 関 数 isalnum, isprint, iskana, iskpun, iskmoji

**isupper, islower, isalpha, isdigit, isalnum, isgraph,
isprint, isascii, isspace, ispunct, iscntrl,
isxdigit, isodigit, iscsym, iscsymf**

レベル 2

書 式 #include <ctype.h>

```
int isupper(c) ; /* 大文字の判定 */
int islower(c) ; /* 小文字の判定 */
int isalpha(c) ; /* 英文字の判定 */
int isdigit(c) ; /* 10 進数の判定 */
int isalnum(c) ; /* 英数字の判定 */
int isgraph(c) ; /* 空白以外の表示可能文字の判定 */
int isprint(c) ; /* 空白を含む表示可能文字の判定 */
int isascii(c) ; /* ASCII 文字の判定 */
int isspace(c) ; /* 空白文字の判定 */
int ispunct(c) ; /* 句読点の判定 */
int iscntrl(c) ; /* 制御文字の判定 */
int isxdigit(c) ; /* 16 進数の判定 */
int isodigit(c) ; /* 8 進数の判定 */
int iscsym(c) ; /* C 言語の識別子として使用できる文字の判定 */
int iscsymf(c) ; /* C 言語の識別子の先頭文字として使用できる文字の判定 */
int c ;
```

機 能

これらの関数は c で与えられた文字の判定を行います。

なお、これらにはマクロ定義と関数版があります。

関数版は ctype.h を、マクロ版は ctype.h をインクルードして使用します。

それぞれ、c が以下の範囲内であるか否かを判定します。

isupper	英大文字	'A'~'Z' (0x41~0x5A)
islower	英小文字	'a'~'z' (0x61~0x7A)
isalpha	英文字	'A'~'Z' (0x41~0x5A) 'a'~'z' (0x61~0x7A)
isdigit	数字	'0'~'9' (0x30~0x39)
isalnum	英文字、数字	'0'~'9' (0x30~0x39) 'A'~'Z' (0x41~0x5A) 'a'~'z' (0x61~0x7A)
isgraph	英文字、記号、数字 (空白を除く)	(0x21~0x7E)
isprint	英文字、記号、数字 (空白を含む)	(0x20~0x7E)
isascii	ASCII 文字	(0x00~0x7F)

isspace	空白文字	(0x09~0x0D、0x20)
ispunct	句読点	(0x20~0x2F)
		(0x3A~0x40)
		(0x5B~0x60)
		(0x7B~0x7E)
iscntrl	制御文字	(0x00~0x1F、0x7F)
isxdigit	16 進数表記用文字	'0'~'9' (0x30~0x39)
		'A'~'F' (0x41~0x46)
		'a'~'f' (0x61~0x66)
isodigit	8 進数表記用文字	'0'~'7' (0x30~0x37)
iscsym	英文字	'A'~'Z' (0x41~0x5A)
		'a'~'z' (0x61~0x7A)
	数字	'0'~'9' (0x30~0x39)
	アンダーライン	'_' (0x5F)
iscsymf	英文字	'A'~'Z' (0x41~0x5A)
		'a'~'z' (0x61~0x7A)
	アンダーライン	'_' (0x5F)

戻り値 条件を満たせば 0 以外の値を、満たさなければ 0 を返します。

参照関数 toascii, tolower, toupper

プログラム例

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

main()
{
    int c;

    for (c = 0x00; c < 0x80; c++) {
        printf("0x%04x", c);
        printf("%3s", isalnum(c) ? "AN" : "");
        printf("%3s", isalpha(c) ? "A" : "");
        printf("%3s", isascii(c) ? "AS" : "");
        printf("\n");
    }
}
```

isatty

レベル 2

書 式 #include <io.h>

```
int isatty(device) ;
int device; /* ファイルハンドル */
```

機 能 deviceで指定したファイルハンドルが文字デバイス（ターミナル、コンソール、プリンタ、シリアルポートなど）であるかどうかを調べます。

戻 り 値 ファイルハンドルが文字デバイスの場合には0以外の値を返し、そうでなければ0を返します。

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <io.h>
#include <stat.h>

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

    file = (argc > 1) ? argv[1] : "con";
    if ((fn = open(file, O_RDONLY)) == -1) {
        perror(file);
        exit(1);
    }

    printf("%s is ", file);
    if (isatty(fn) == 0)
        printf("not character device\n");
    else
        printf("character device\n");

    close(fn);
    exit(0);
}
```

iskana, iskpun, iskmoji

レベル 2

書 式 #include <jfctype.h>

```
int iskana(c) ; /* カナ文字またはカナ句読点の判定 */
int iskpun(c) ; /* カナ句読点の判定 */
int iskmoji(c) ; /* カナ文字の判定 */
int c ; /* 判定する文字 */
```

機 能 これらの関数はcで与えられた文字の判定を行います。

iskana はcがカナ文字またはカナ句読点であるか否かを判定します。

iskpun はcがカナ句読点であるか否かを判定します。

iskmoji はcがカナ文字であるか否かを判定します。

戻 り 値 条件を満たせば0以外の値を、満たさなければ0を返します。

参 照 関 数 isalpha, islower, isupper, isxdigit, isdigit, ispunct, isspace, isalnum, isalkana, ispnkana, isalnmkana, isprkana, zentohan

プログラム例

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

main()
{
    int c;

    /* 1byteカナ文字だけ出力するフィルタ */
    while ((c = getchar()) != EOF) {
        if (iskanji(c))
            (void)getchar();
        else if (iskana(c))
            putchar(c);
    }
}
```

iskanji, iskanji2

レベル 2

書 式 #include <jfctype.h>

```
int iskanji(c) ; /* 漢字の1バイト目の判定 */
int iskanji2(c) ; /* 漢字の2バイト目の判定 */
int c ; /* 判定する文字 */
```

機 能 これらの関数はcで与えられた文字の判定を行います。

iskanji はcがシフト JIS コードの1バイト目であるか否かを判定します。

iskanji2 はcがシフト JIS コードの2バイト目であるか否かを判定します。

シフト JIS コードの1バイト目…… $0x81 \leq c \leq 0x9F$ または $0xE0 \leq c \leq 0xEF$

シフト JIS コードの2バイト目…… $0x40 \leq c \leq 0x7F$ または $0x80 \leq c \leq 0xFC$

戻 り 値 条件を満たせば0以外の値を、満たさなければ0を返します。

参 照 関 数 jiszen, jis10, jis11, jis12



itoa

レベル 2

書 式 `#include <stdlib.h>`

```
char *itoa(value, string, radix);  
int value; /* 変換すべき整数値 */  
char *string; /* 変換した文字列を格納する領域を指すポインタ */  
int radix; /* 基数 */
```

機 能 value で指定した整数値を文字列（最大 33 バイト）に変換し、string の指す領域に格納します。

文字列の末尾にはヌル文字‘\0’が付加されます。

radix は value の基数で、2～36 の範囲で進数を指定します。

もし radix が 10 で value の値が負の場合は、文字列の先頭にマイナス記号（-）が付加されます。

戻 り 値 変換された文字列を指すポインタを返します。

参 照 関 数 ltoa, ultoa, wtoa

プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
  
main()  
{  
    char    buf[34];  
    int      i;  
  
    for (i = 0; i < 100; i++) {  
        printf("%2d : ", i);  
        printf("%7s ", itoa(i, buf, 2));  
        printf("%3s ", itoa(i, buf, 8));  
        printf("%2s ", itoa(i, buf, 10));  
        printf("%2s ", itoa(i, buf, 16));  
        printf("%n");  
    }  
}
```

jisalpha, jisupper, jislower, jisdigit, jiskata, jishira, jiskigou, jisspace

レベル 2

書 式 #include <jfctype.h>

```
int jisalpha(c) ; /* 全角英文字の判定 */
int jisupper(c) ; /* 全角英大文字の判定 */
int jislower(c) ; /* 全角英小文字の判定 */
int jisdigit(c) ; /* 全角数字の判定 */
int jiskata(c) ; /* 全角カタカナ文字の判定 */
int jishira(c) ; /* 全角ひらがな文字の判定 */
int jiskigou(c) ; /* 全角句読点文字の判定 */
int jisspace(c) ; /* 全角空白文字の判定 */
int c ; /* 判定する文字 */
```

機 能 これらの関数はcで与えられたシフト JIS コードでの文字の判定を行います。

jisalpha はcが全角英文字であるか否かを判定します。

jisupper はcが全角英大文字であるか否かを判定します。

jislower はcが全角英小文字であるか否かを判定します。

jisdigit はcが全角数字であるか否かを判定します。

jiskata はcが全角カタカナ文字であるか否かを判定します。

jishira はcが全角ひらがな文字であるか否かを判定します。

jiskigou はcが全角句読点文字であるか否かを判定します。

jisspace はcが全角空白文字であるか否かを判定します。

戻 り 値 条件を満たせば0以外の値を、満たさなければ0を返します。

参 照 関 数 jiszen, jislo, jisll, jis2, iskanji, iskanji2

jiszen, jis0, jis1, jis2

レベル 2

書 式 #include <jfctype.h>

```
int jiszen(c) ; /* 全角文字の判定 */
int jis0(c) ; /* 全角空白文字、JIS 第一水準記号の判定 */
int jis1(c) ; /* JIS 第一水準漢字の判定 */
int jis2(c) ; /* JIS 第二水準漢字の判定 */
int c ; /* 判定する文字 */
```

機

能

これらの関数は c で与えられたシフト JIS コードでの文字の判定を行います。

jiszen は c が全角文字であるか否かを判定します。

jis0 は全角空白文字または JIS 第一水準の記号（非漢字）であるか否かを判定します。

jis1 は c が JIS 第一水準漢字であるか否かを判定します。

jis2 は c が JIS 第二水準漢字であるか否かを判定します。

戻 り 値

条件を満たせば 0 以外の値を、満たさなければ 0 を返します。

参 照 関 数

iskanji, iskanji2

プログラム例

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

main()
{
    unsigned short  ch;
    int             c;
    int             cc;

    while ((c = getchar()) != EOF) {
        if (iskanji(c)) {
            cc = c;
            ch = (c << 8);
            ch += (c = getchar());
            printf("%c%c は、", cc, c);
            if (jis0(ch))
                printf("非漢字\n");
            else if (jis1(ch))
                printf("第1水準\n");
            else if (jis2(ch))
                printf("第2水準\n");
            else
                printf("1byte文字\n");
        }
    }
}
```


jstrchr

レベル 2

書 式 #include <jstring.h>

```
unsigned char *jstrchr(string, c);
unsigned char *string; /* サーチが行われる文字列へのポインタ */
int c; /* サーチする文字 */
```

機 能 string が指す文字列の中で、文字 c が最初に現れる位置を調べます。

本関数は、strchr 関数の漢字対応版であり、文字列にシフト JIS コードの漢字を含むことができるという点を除けば、基本的な機能は同じです。

漢字は 1 文字として扱われ、シフト JIS コードの 2 バイト目がヌル文字 '\0' であれば 1 バイト目もヌル文字であると解釈されます。

サーチする文字が漢字の場合は、シフト JIS コードを使用し、このコードの 1 バイト目を上位バイトに、2 バイト目を下位バイトに設定します。

サーチする文字が半角の場合には、下位 8 ビットを使用し、上位バイトはゼロにしておきます。

戻 り 値 文字 c が現れた位置を指すポインタを返します。

文字 c が文字列にないときは NULL を返します。

参 照 関 数 strchr, jstrrchr

jstrcmp

レベル 2

書 式 #include <jstring.h>

```
int jstrcmp(string1, string2) ;  
unsigned char * string1; /* 文字列 1 へのポインタ */  
unsigned char * string2; /* 文字列 2 へのポインタ */
```

機 能

本関数は strcmp 関数の漢字対応版であり、文字列にシフト JIS コードの漢字を含むことができるという点を除けば、基本的な機能は同じです。

漢字は 1 文字として扱われ、シフト JIS コードの 2 バイト目がヌル文字'¥0'であれば 1 バイト目もヌル文字であると解釈されます。

なお、文字列の比較は次の大小関係によって行われます。

英数文字<カナ文字<漢字

戻 り 値

文字列 1 と文字列 2 の大小関係を返します。

戻り値とその内容は次の通りです。

戻り値	内 容
負	string1 は string2 より小さい
0	string1 と string2 は等しい
正	string1 は string2 より大きい

参 照 関 数

strcmp

jstrncmp

レベル 2

書 式 #include <jstring.h>

```
int jstrncmp(string1, string2, n) ;
/* unsigned char * string1; /* 文字列 1 へのポインタ */
/* unsigned char * string2; /* 文字列 2 へのポインタ */
/* int n; /* 比較する文字数 */
```

機 能 文字列の比較を行います。

本関数は strncmp 関数の漢字対応版であり、文字列にシフト JIS コードの漢字を含むことができるという点を除けば、基本的な機能は同じです。

n は比較する文字数を指定するのでありバイト数を指定するものではありません。

したがって、漢字は 1 文字として数えます。

また、シフト JIS コードの 2 バイト目がヌル文字 '\0' ならば、1 バイト目もヌル文字であると解釈します。

なお、jstrncmp 関数は、次の大小関数により比較を行います。

英数文字 < カナ文字 < 漢字

戻 り 値 文字列 1 と文字列 2 の大小関係を返します。

戻り値とその内容は次の通りです。

戻り値	内 容
負	string1 は string2 より小さい
0	string1 と string2 は等しい
正	string1 は string2 より大きい

参 照 関 数 strncmp

jstrchr

レベル 2

書 式 #include <jstring.h>

```
unsigned char * jstrchr(string, c);  
unsigned char * string; /* サーチが行われる文字列へのポインタ */  
int c; /* サーチする文字 */
```

機 能

string1 が指す文字列の中で、文字 c が最後に現れた位置を調べます。

本関数は、strrchr 関数の漢字対応版であり、文字列にシフト JIS コードの漢字を含むことができるという点を除けば、基本的な機能は同じです。

漢字は 1 文字として扱われ、シフト JIS コードの 2 バイト目がヌル文字 '¥0' であれば 1 バイト目もヌル文字であると解釈されます。

サーチする文字が漢字の場合は、シフト JIS コードを使用し、このコードの 1 バイト目を上位バイトに、2 バイト目を下位バイトに設定します。

サーチする文字が半角の場合には、下位 8 ビットを使用し、上位バイトはゼロにします。

戻 り 値

文字 c が最後に現れた位置を指すポインタを返します。

文字 c が文字列にないときは NULL を返します。

参 照 関 数

strrchr, strpbrk, jstrchr

kbhit

レベル 2

書 式 #include <conio.h>

int kbhit () ;

機 能 コンソールからの入力があるかどうかを調べます。
入力を調べるのみで、入力はいりません。

戻 り 値 入力があるときは0以外を返し、ないときは0を返します。

プログラム例

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

main()
{
    puts("何かキーを押してください。");
    while (kbhit() == 0)
        ;
}
```

key

レベル 3

書 式 #include <basic0. h>

```
void key(n, s) ;  
int n; /* ファンクションキー番号 */  
char *s; /* 文字列 */
```

機 能

再定義可能なファンクションキーに文字列を定義します。

n にはファンクションキーの番号を指定します。(1~20)。

s には文字列を指定します。

ファンクションキーに定義できる文字列の長さは 31 バイトまで有効です。

定義内容に 32 バイト以上を指定した場合はエラーになり、何もしません。

BASIC から C に変換したときに KEY 文が key 関数に展開されますが、BASIC の KEY 文とは動作が異なりますので注意してください。

制御コードは、@ を前につけた A~Z, [, ¥,], ^, _ (¥は¥¥にすること) で表します。(例. “@A”)。

“@@”は引用符 (”) を表します。

戻 り 値

戻り値はありません。

keysns

レベル 3

書 式 #include <image.h>

int keysns();

機 能 キーボードの入力状態を調べます。

戻 り 値 入力状態によって、以下の値を返します。

入力あり : -1

入力なし : 0

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

main()
{
    double x;
    int i;
    int j;

    for (i = -32; i <= 32; i++)
        for (j = -32; j <= 32; j++)
            x = (double)i / 64.0;
    printf("index(%d,%d) = %f\n",
           x, j, index(x, j));
}
```

ldexp

レベル 2

書 式 `#include <math.h>`

```
double ldexp(x, exp);  
double x;  
int exp;
```

機 能 $x \times 2^{\text{exp}}$ を求めます。

戻 り 値 $x \times 2^{\text{exp}}$ を計算した結果を返します。

オーバーフローが発生した場合は、errnoにERANGEがセットされ、返り値は、xと同じ符号がついたHUGE_VALになります。

アンダーフローが発生した場合は、errnoにERANGEがセットされ、返り値は、xと同じ符号がついた0になります。

参 照 関 数 frexp, modf

プログラム例

```
#include <stdio.h>  
#include <math.h>  
  
main()  
{  
    double x;  
    int i;  
    int j;  
  
    for (i = -32; i <= 32; i++) {  
        for (j = -32; j <= 32; j++) {  
            x = (double)i / 10.0;  
            printf("ldexp(%+4.1f, %d) = %g\n",  
                x, j, ldexp(x, j));  
        }  
    }  
}
```


ldiv

レベル 2

書 式 #include <stdlib.h>

```
ldiv_t ldiv(number, denom) ;
long number; /* 割られる値 */
long denom; /* 割る値 */
```

機 能

number を denom で割った商と余りを ldiv_t 型の構造体として返します。

ldiv は符号付き整数の除算を行う場合に、処理系の違いによる結果の相違を解消するために用意されています。

ldiv_t 型構造体は、符号付き整数2つをメンバに持つ構造体で、stddef.h で以下のように型宣言されています。

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

戻 り 値

number を denom で割った商と余りを ldiv_t 型の構造体として返します。

戻り値は、構造体の実体であることに注意してください。

構造体メンバ quot には商を、メンバ rem には余りを返します。

参 照 関 数 div

プログラム例

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

main()
{
    ldiv_t  ans;
    long    num;
    long    denom;

    for (num = -10L; num <= 10L; num++) {
        for (denom = -10L; denom <= 10L; denom++){
            if (denom == 0L)
                continue;
            ans = ldiv(num, denom);
            printf("%ld / %ld = %ld 余り %ld\n",
                num, denom, ans.quot, ans.rem);
        }
    }
}
```

書 式 #include <graph.h>

```
void line (x1, y1, x2, y2, p, ls) ;
int x1; /* 始点 X 座標 */
int y1; /* 始点 Y 座標 */
int x2; /* 終点 X 座標 */
int y2; /* 終点 Y 座標 */
int p; /* パレットコード */
int ls; /* ラインスタイル */
```

機 能 グラフィック画面に直線を描画します

x1 には始点 X 座標を指定します (−32768~32767)。

y1 には始点 Y 座標を指定します (−32768~32767)。

x2 には終点 X 座標を指定します (−32768~32767)。

y2 には終点 Y 座標を指定します (−32768~32767)。

p にはパレットコードを指定します (0~65535)。

画面モードにより最大値が異なります。

ls にはラインスタイルを指定します (0~65535)。

戻 り 値 戻り値はありません。

参 照 関 数 screen

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

main()
{
    long num;
    long denom;
    long ans;

    for (num = -100; num < 100; num++)
        for (denom = -100; denom < 100; denom++)
            if (denom < 0)
                continue;
            ans = div(num, denom);
            printf("%ld \ %ld = %ld \ %ld\n",
                num, denom, ans.quot, ans.rem);
}
```

プログラム例

```

#include      <stdlib.h>
#include      <basic0.h>
#include*     <graph.h>

main()
{
    int      i;
    int      x1, x2, y1, y2;
    int      pc;
    int      ls;

    b_init();
    screen(2, 0, 1, 1);
    vpage(1);
    for (i = 0; i < 500; i++) {
        x1 = rand() % 768;
        y1 = rand() % 512;
        x2 = rand() % 768;
        y2 = rand() % 512;
        pc = rand() % 16;
        ls = rand() * 2 + 1;
        line(x1, y1, x2, y2, pc, ls);
    }
    b_exit(0);
}

```

実行

参照関係

localtime

レベル 2

書 式 #include <time.h>

```
struct tm *localtime(time);  
time_t *time; /* 時刻データの格納領域を指すポインタ */
```

機 能

時刻 time を地方時に変換し、tm 構造体に格納します。
時刻 time はグリニッジ標準時 1970 年 1 月 1 日 0 時 0 分 0 秒から経過した秒数で、time 関数を呼び出すことにより得られます。
tm 構造体は time.h で以下のように宣言されています。

```
struct tm {  
    int  tm_sec; /* 秒(0~59) */  
    int  tm_min; /* 分(0~59) */  
    int  tm_hour; /* 時間(0~23) */  
    int  tm_mday; /* 日付(1~31) */  
    int  tm_mon; /* 月(0~11、0 が 1 月) */  
    int  tm_year; /* 年(現在の年から 1900 を引いた数) */  
    int  tm_wday; /* 曜日(0~6、0 が 日曜日) */  
    int  tm_yday; /* 1 年を通した日付(0~365、0 は 1 月 1 日) */  
    int  tm_isdst; /* 夏時間調整が無効の場合は 0、  
                  有効の場合は 0 以外 */  
};
```

localtime は、内部の静的な領域へのポインタを返します。
したがって、localtime が呼び出されるたびに同じ領域に上書きされ、前の内容は失われます。
それ以外にも、ctime 関数を呼び出すと、ctime 関数の内部で localtime を呼び出すので、それ以前に呼び出された、localtime で返された内容は失われます。

戻 り 値 tm 構造体を指すポインタを返します。

参 照 関 数 asctime, ctime, gmtime, time

locate

レベル 3

書 式 #include <basic0. h>

```
void locate(x, y);
int x; /* X座標 */
int y; /* Y座標 */
```

機 能 テキスト座標の x, y で指定した位置にカーソルを移動します。

戻 り 値 戻り値はありません。

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

main()
{
    double x;
    int i;

    for (i = 1; i <= 30; i++) {
        x = (double) i / 10.0;
        printf("log(x+4.11) = %f\n", x, log(x));
    }

    for (i = 1; i <= 30; i++) {
        x = (double) i / 10.0;
        printf("log10(x+4.11) = %f\n", x, log10(x));
    }
}
```

log, log10

レベル 2

書 式 #include <math.h>

```
double log(x) ; /* 自然対数 */
double log10(x) ; /* 底が 10 の対数 */
double x ;
```

機 能 log は x の e を底とする自然対数を、log10 は x の 10 を底とする常用対数を求めます。

戻 り 値 log は $\log_e X$ を、log10 は $\log_{10} X$ を返します。
引数が負の数などの不正な値の場合は errno に EDOM がセットされ、返り値は不定になります。
またエラー処理は、matherr 関数を使って行うことができます。

参 照 関 数 exp, matherr, pow

プログラム例

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

main()
{
    double x;
    int i;

    for (i = 1; i <= 30; i++) {
        x = (double)i / 10.0;
        printf("log(%+4.1f) = %g\n", x, log(x));
    }
    for (i = 1; i <= 30; i++) {
        x = (double)i / 10.0;
        printf("log10(%+4.1f) = %g\n", x, log10(x));
    }
}
```

longjmp

レベル 2

書 式 #include <setjmp. h>

```
void longjmp(env, value) ;  
jmp_buf env; /* setjmp によりセーブされたスタック環境 */  
int value; /* setjmp が返す (かのように見える) 戻り値 */
```

機 能 setjmp 関数でセーブしたスタック環境を復元して、対応する setjmp 関数の呼び出し元へ制御を移します。

env は setjmp. h で定義された jmp_buf 型の変数であり、あらかじめ setjmp 関数によってスタック環境がセーブされていなければなりません。

value は、setjmp 関数の呼び出し元へ、setjmp 関数の戻り値として渡される値です。

必ず 0 以外の値を指定してください。

もし 0 を指定すると、強制的に 1 を指定したことになります。

戻 り 値 longjmp は、setjmp 関数を呼び出す前に呼び出してはいけません。

また、対応する setjmp 関数の呼び出し元の関数が return 文を実行した後や、関数から抜け出したりした後で longjmp をしてはいけません。

ほかに、setjmp 関数呼び出し時と longjmp 呼び出し時でスーパーバイザ/ユーザーモードが異なっていてはいけません。

もし、上記の禁止事項が守られていない状態で longjmp を行うと、その後のプログラムの動作は保障されません。

setjmp 関数の呼び出し元へ制御を移すので、longjmp の呼び出し元へはリターンしません。

setjmp 関数を呼び出すルーチンにおけるレジスタ変数は、longjmp を実行した後では、正しい値として使用することはできません。

参 照 関 数 setjmp

書 式 #include <io.h>

```
long lseek(handle, offset, origin) ;
int handle; /* ファイルのファイルハンドル */
long offset; /* 起点からのオフセット (移動バイト数) */
int origin; /* ファイルポインタの起点 */
```

機 能 handle で指定されたファイルのファイルポインタを origin で指定した起点から offset バイト分移動します。
origin の値と内容を以下に示します。

origin の値	内 容
SEEK_SET	ファイルの先頭
SEEK_CUR	ファイルポインタの現在の位置
SEEK_END	ファイルの最後

ファイルの先頭よりも前へ移動することはできません。
ファイルの最後より後へ移動すると、ファイルの最後の位置から移動後の位置まで、ファイルに 0 を書き足します。

戻 り 値 新しいファイルポインタの位置を先頭からのバイト数で返します。
エラーが発生した場合は -1 を返し、errno に次の値がセットされます。

errno の値	内 容
EBADF	ファイルが無効
EINVAL	origin の値が無効、または offset で指定された位置がファイルの先頭位置よりも前を示している

シークが不可能なデバイス（ターミナルやプリンタなど）では、戻り値は不定となります。

参 照 関 数 fseek, tell

プログラム例

```

#include      <stdio.h>
#include      <stdlib.h>
#include      <fcntl.h>
#include      <io.h>
#include      <stat.h>

#define LEN      100L

main(int argc, char *argv[])
{
    char      buf[LEN];
    char      *file;
    int      fn;

    file = (argc > 1) ? argv[1] : "file";
    if ((fn = open(file, O_RDONLY|O_BINARY)) == -1) {
        perror(file);
        exit(1);
    }
    if (lseek(fn, -LEN, SEEK_END) == -1L) {
        perror("seek error");
        exit(1);
    }
    else {
        read(fn, buf, LEN);
        write(1, buf, LEN);
    }
    close(fn);
    exit(0);
}

```

ltoa

レベル 2

書 式 #include <stdlib.h>

```
char *ltoa(value, string, radix);  
long value; /* 変換すべき long 型整数値 */  
char *string; /* 変換した文字列を格納する領域を指すポインタ */  
int radix; /* 基数 */
```

機 能

value で指定された long 型整数値を文字列（最大 33 バイト）に変換し、string が指す領域に格納します。

文字列の末尾にはヌル文字 '\0' が付加されます。

radix は value の基数で、2 から 36 の範囲で進数を指定します。

もし radix が 10 で value が負の場合は、文字列の先頭にマイナス記号（-）が付加されます。

戻 り 値

変換された文字列を指すポインタを返します。

参 照 関 数

itoa, ultoa, wtoa

プログラム例

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

```
main()
```

```
{
    char    buf[34];
    long    l;
    int     i;
```

```
    for (i = -10; i < 10; i++) {
        l = (long)i * 10000L;
        printf("%7ld: ", l);
        printf("%032s ", ltoa(l, buf, 2));
        printf("%11s ", ltoa(l, buf, 8));
        printf("%7s ", ltoa(l, buf, 10));
        printf("%8s ", ltoa(l, buf, 16));
        printf("%n");
    }
```

```
}
```

malloc

レベル 2

書 式 #include <stdlib.h>

```
void *malloc(size) ;  
size_t size; /* 割りつけるメモリ領域のサイズ */
```

機 能 size バイトのメモリ領域を割りつけます。

戻 り 値 割りつけた領域を指すポインタを返します。
割りつけられた領域は、どんな型のデータにも利用できるようになっています。
メモリに空きがないときは NULL を返します。

参 照 関 数 calloc, free, realloc

プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
  
#define B_SIZE 1024  
  
main()  
{  
    int i;  
    char *p;  
  
    for (i = 1; i < 1024; i++) {  
        printf("%4d block ", i);  
        if ((p = malloc(i * B_SIZE)) == NULL) {  
            printf("確保できませんでした。¥n");  
            break;  
        }  
        printf("確保しました。");  
        free(p);  
        printf("; 解放しました。¥n");  
    }  
    exit(0);  
}
```

m_alloc

レベル 3

書 式 #include <music2, h>

```
int m_alloc(t, size) ;
char t; /*  トラック番号   */
int size; /*  トラックサイズ   */
```

機 能 FM 音源および MIDI のためのトラックバッファを確保します。
全トラックバッファ内のデータはすべてクリアされます。
t にはトラック番号を指定します (1~223)。
size にはトラックサイズをバイト数で指定します (1~65536)。

戻 り 値 正常終了の場合は 0 を返します。
引数のエラーなどの場合エラー番号として、0 以外の正の数を返します。
FM 音源が使用不可の場合、-1 を返します。
なお、エラー番号とその意味は以下の通りです。

番号	エラーの意味
-1	音楽用関数は無効です
1	テンポの指定が無効です
2	トラック番号が無効です
3	music cmd error
4	サイズの指定が無効です
5	メモリー確保できません
6	チャンネル番号が無効です
7	配列の指定に誤りがあります
8	音色番号が無効です
9	音楽用関数は無効です
19	MML の文法に誤りがあります
20	[] の中の指定に誤りがあります
21	] がありません
22	繰り返しの値が無効です
23	繰り返し番号の指定がありません
24	繰り返し番号の指定が無効です
25	オクターブの番号がありません
26	オクターブの指定が無効です

以上のエラー番号の説明は、以下の関数においても同様です。

344

m_assign

レベル 3

書 式 #include <music2.h>

```
int m_assign(channel, track) ;
char channel; /* FM 音源のチャンネル */
char track; /* トラック番号 */
```

機 能 FM 音源または MIDI のチャンネルにトラックを割りあてます。

デフォルトでは、1~8 チャンネルがそれぞれ 1~8 トラックに対応しています。

また、MIDI ボードが本体に実装されている場合、9~24 チャンネルはそれぞれデフォルトで 81~96 トラックに対応しています。

9~24 のチャンネルは MIDI ボードのためのチャンネルで、MIDI ボードがなければ使用できません。

戻 り 値 正常終了の場合は 0 を返します。

引数のエラーなどの場合エラー番号として 0 以外の正の数を返します。

FM 音源が使用不可の場合、-1 を返します。

なお、エラー番号とその意味は、m_alloc 関数を参照してください。

型	内容
DOMAIN	指定の範囲が正しい
SING	引数が正しい
OVERFLOW	オーバーフローした
UNDERFLOW	アンダーフローした
LOSS	同時に超過した
PLUS	同時に超過した

matherr

レベル 2

書 式 #include <math.h>

```
int matherr(x);  
struct exception *x; /* exception 構造体を指すポインタ */
```

機 能

数値演算ライブラリの実行時に発生したエラーを処理します。

数値演算関数でエラーが発生すると、必ず `matherr` が呼び出されるので、ユーザーは独自のエラー処理を行うことができます。

`matherr` は、`math.h` で以下のように宣言されている構造体 `exception` へのポインタを引数に持って呼び出されます。

```
struct exception {  
    int type; /* エラーの種類 */  
    char *name; /* エラーの発生した関数名を指すポイン  
                タ */  
    double arg1, arg2, retval; /* argn はエラーを起した引数  
                                retval は戻り値 */  
};
```

`type` は数値演算時に発生したエラーの種類を示します。

type の値	内 容
DOMAIN	引数の領域が正しくない
SING	引数が正しくない
OVERFLOW	オーバーフローした
UNDERFLOW	アンダーフローした
TLOSS	全体的に桁落ちした
PLOSS	部分的に桁落ちした

`name` はエラーをおこした関数名を指すポインタです。

関数名はヌル文字で終わっています。

`arg1` と `arg2` はエラーを起した引数の値を示しています (引数が1つの場合は `arg1` に格納されます)。

`retval` はエラーが発生した数値演算関数のデフォルトの戻り値で、ユーザーがここに数値を代入して、それを変更することもできます。

matherr 自身の戻り値は、エラーが実際に起ったかどうかを指示しなければなりません。

matherr が 0 を返した場合は、errno に対応するエラー値をセットします。

matherr が 0 以外の値を返した場合には、errno は変化しません。

このとき、retval に数値を代入して、数値演算関数の戻り値を変更できます。

戻り値 エラー回復できなかった場合には 0 を返し、エラー回復を行うことができた場合には 0 以外の値を返さなければなりません。

参照関数 acos, asin, atan, atan2, cabs, cos, cosh, exp, hypot, log, pow, sin, sinh, sqrt, tan

プログラム例

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

main()
{
    int matherr(struct exception *);
    double x, y;

    x = -10.0;
    y = log(x);
}

static int matherr(struct exception *ep)
{
    char *s;
    switch (ep->type) {
    case DOMAIN:
        s = "引数の領域が正しくない。";
        break;
    case SING:
        s = "引数が正しくない。";
        break;
    case OVERFLOW:
        s = "オーバーフローした。";
        break;
    case UNDERFLOW:
        s = "アンダーフローした。";
        break;
    case TLOSS:
        s = "全体的に桁落ちした。";
        break;
    case PLOSS:
        s = "部分的に桁落ちした。";
        break;
    default:
        return (0);
    }
    fprintf(stderr, "%s: %s\n", ep->name, s);
    return (0);
}
```

max

レベル 2

書 式 `#include <stdlib.h>`

`(type)max(a, b);`

`(type)a;`

`(type)b;`

機 能

max は、a と b を比較し、大きい方を返すマクロです。

2つの引数と戻り値の型は同じ型で、比較演算が可能な型でなければなりません。

引数には、副作用を引き起こすものや、関数などを指定すると、期待した結果と異なる場合がありますので注意して下さい。

戻 り 値

2つの引数のうち、大きい方の値を返します。

参 照 関 数

min

プログラム例

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

#define SIZE 10

static int a[SIZE] = {3, 1, 4, 1, 5, 9, 2, 5, 7, 1};

main()
{
    int m;
    int i;

    m = a[0];
    for (i = 0; i < SIZE; i++)
        m = max(m, a[i]);
    printf("最大の値は%d\n", m);
}
```

m_cont

レベル 3

書 式 #include <music2.h>

```
int m_cont(c1, c2, c3, c4, c5, c6, c7, c8) ;
int c1, c2, c3, c4, c5, c6, c7, c8; /* チャンネル */
```

機 能 FM 音源のチャンネルを指定して、一時中断していた演奏を再開します。
指定するチャンネルの個数が7個以下の場合、余りの引数に 'NASI' を指定しなければなりません。
全チャンネルが 'NASI' ならば、1~8 チャンネル中現在中断していたチャンネルすべてを再開します。
チャンネルは1~8が指定できます。

戻 り 値 正常終了の場合は0を返します。
引数のエラーなどの場合、エラー番号として0以外の正の数を返します。
FM 音源が使用不可の場合、-1を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

md_cont

レベル 3

書 式 `#include <music2.h>`

```
int md_cont(channel) ;  
int channel ;    /* ビット指定のチャンネル */
```

機 能 FM 音源および MIDI のチャンネルをビットで指定して、一時中断していた演奏を再開します。

channel に設定する値は、以下の通りです。

ビット 0～7 : 1～8 チャンネル (FM 音源)

ビット 8～23 : 9～24 チャンネル (MIDI)

ビット 24～31 : 常に 0

ビットを 1 にしたチャンネルの演奏を再開します。

0 または 'NASI' を指定すると、現在中断していたチャンネルすべてを再開します。

戻 り 値 正常終了の場合は 0 を返します。

引数のエラーなどの場合、エラー番号として 0 以外の正の数を返します。

FM 音源を使用不可の場合、-1 を返します。

なお、エラー番号とその意味は、m_alloc 関数を参照してください。

md_init

レベル 3

書 式 #include <music2.h>

```
int md_init() ;
```

機 能 MIDI ボートの初期化のみを行います。

m_assign で割りあてたトラックは、m_init で初期化してください。

MIDI ボードを使用する場合には、必ず m_init の実行直後に実行してください。

MIDI ボードを使用しない場合には、実行する必要はありません。

MIDI ボードがない場合には、エラーになります。

戻 り 値 正常終了の場合は 0 を返します。

MIDI ボードがない場合、エラー番号 49 を返します。

FM 音源が使用不可の場合、-1 を返します。

なお、エラー番号とその意味は、m_alloc 関数を参照してください。

参照関数 m_alloc, m_init

md_off

レベル 3

書 式 #include <music2.h>

```
int md_off(channel) ;  
int channel ; /* ビット指定のチャンネル */
```

機 能 MIDI のチャンネルをビットで指定して、MIDI ボードへの出力を停止します。
channel に設定する値は、以下の通りです。

ビット0～7 : 常に 0

ビット8～23 : 9～24 チャンネル (MIDI)

ビット24～31 : 常に 0

ビットを 1 にしたチャンネルの出力を停止します。

0 または 'NASI' を指定すると、9～24 すべてのチャンネルの出力を停止します。

MIDI ボードがない場合には、エラーになります。

戻 り 値 正常終了の場合は 0 を返します。

引数のエラーなどの場合、エラー番号として 0 以外の正の数を返します。

FM 音源が使用不可の場合、-1 を返します。

なお、エラー番号とその意味は、m_alloc 関数を参照してください。

md_on

レベル 3

書 式 #include <music2.h>

```
int md_on(channel) ;
int channel ; /* ビット指定のチャンネル */
```

機 能 MIDI のチャンネルをビットで指定して、MIDI ボードへの出力を開始します。
channel に設定する値は、以下の通りです。

ビット0～7 : 常に 0
ビット8～23 : 9～24 チャンネル (MIDI)
ビット24～31 : 常に 0

ビットを 1 にしたチャンネルの出力を開始します。

0 または 'NASI' を指定すると、9～24 すべてのチャンネルの出力を開始します。
MIDI ボードがない場合には、エラーになります。

戻 り 値 正常終了の場合は 0 を返します。
引数のエラーなどの場合、エラー番号として 0 以外の正の数を返します。
FM 音源が使用不可の場合、-1 を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

md_play

レベル 3

書 式 #include <music2.h>

```
int md_play(channel) ;  
int channel ; /* ビット指定のチャンネル */
```

機 能 FM 音源および MIDI のチャンネルをビットで指定して、演奏を開始します。
channel に設定する値は、以下の通りです。

ビット0～7 : 1～8 チャンネル (FM 音源)

ビット8～23 : 9～24 チャンネル (MIDI)

ビット24～31 : 常に 0

ビットを 1 にしたチャンネルの演奏を開始します。

0 または 'NASI' を指定すると、すべてのチャンネルの演奏を開始します。

戻 り 値 正常終了の場合は 0 を返します。

引数のエラーなどの場合、エラー番号として 0 以外の正の数を返します。

FM 音源が使用不可か MIDI ボードがない場合、-1 を返します。

なお、エラー番号とその意味は、m_alloc 関数を参照してください。

md_regr

レベル 3

書 式 #include <music2.h>

```
int md_regr(grpno, regno) ;  
char grpno; /* グループ番号 */  
char regno; /* レジスタ番号 */
```

機 能

MIDI コントローラのレジスタを読み込みます。

MIDI コントローラの grpno で指定したグループの regno で指定したレジスタを読み込みます。

グループ番号 grpno は 0~15 を指定します。

レジスタ番号 regno は 0~15 を指定します。

MIDI ボードがない場合や、存在しないレジスタを指定した場合にはエラーになります。

戻 り 値

正常終了の場合は読み込んだレジスタの内容を返します。

FM 音源が使用不可か MIDI ボードがない場合、-1 を返します。

存在しないレジスタを指定した場合、-2 を返します。

md_regw

レベル 3

書 式 #include <music2.h>

```
int md_regw(grpno, regno, data) ;  
char grpno; /* グループ番号/*  
char regno; /* レジスタ番号 /*/  
char data; /* 書き込むデータ */
```

機 能

MIDI コントローラのレジスタにデータを書き込みます。

MIDI コントローラの grpno で指定したグループの regno で指定したレジスタに data で指定したデータを書き込みます。

グループ番号 grpno は 0~15 を指定します。

レジスタ番号 regno は 0~15 を指定します。

書き込むデータ data は 0~255 を指定します。

MIDI ボードがない場合や、存在しないレジスタを指定した場合には、エラーになります。

戻 り 値

正常終了の場合は、書き込んだデータを返します。

FM 音源が使用不可か MIDI ボードがない場合、-1 を返します。

存在しないレジスタを指定した場合、-2 を返します。

md_stat

レベル 3

書 式 #include <music2.h>

```
int md_stat(ch) ;
int ch; /* チャンネル */
```

機 関 能 ch で指定したチャンネルの MIDI ボードへの出力の状態を調べます。
ch には 9~24 を指定します。
0 または 'NASI' を指定すると、9~24 すべてのチャンネルの状態を調べます。
MIDI ボードがない場合には、エラーになります。

戻 り 値 正常終了の場合はチャンネルの状態を返します。
このとき、ch に指定した値により以下ようになります。

・ ch が 9~24 のとき

0 : 停止中

1 : 出力中

・ ch が 0 または 'NASI' のとき

ビット 0~7 : 常に 0

ビット 8~23 : 9~24 チャンネルの状態
(ビットが 0 で停止中、1 で出力中)

ビット 24~31 : 常に 0

FM 音源が使用不可か MIDI ボードがない場合、-1 を返します。

md_stop

レベル 3

書 式 #include <music2. h>

```
int md_stop(channel) ;  
int channel; /* ビット指定のチャンネル */
```

機 能 FM 音源および MIDI のチャンネルをビットで指定して、演奏を一時中断します。

channel に設定する値は、以下の通りです。

ビット0～7 : 1～8 チャンネル (FM 音源)

ビット8～23 : 9～24 チャンネル (MIDI)

ビット24～31 : 常に 0

ビットを 1 にしたチャンネルの演奏を一時中断します。

0 または 'NASI' を指定すると、現在演奏しているチャンネルすべてを一時中断します。

戻 り 値 正常終了の場合は 0 を返します。

引数のエラーなどの場合、エラー番号として 0 以外の正の数を返します。

FM 音源が使用不可の場合、-1 を返します。

なお、エラー番号とその意味は、m_alloc 関数を参照してください。

md_wrt

レベル 3

書 式 #include <music2. h>

```
int md_wrt(data) ;
char data; /* 出力データ */
```

機 能 MIDI ボードにデータを出力します。
MIDI ボードに data で指定したデータを出力します。
出力データ data は 0~255 を指定します。
MIDI ボードがない場合には、エラーになります。

戻 り 値 正常終了の場合は 0 を返します。
引数のエラーなどの場合、エラー番号として 0 以外の正の数を返します。
FM 音源が使用不可の場合、-1 を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

関 数 関 数

関 数 関 数

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

#define SIZE 256

static char sbuf[SIZE];
static char dbuf[SIZE];

main()
{
    char *p;
    int i;

    p = "hello world good bye";

    if ((p = memcpy(sbuf, p, SIZE)) != NULL)
        printf("sbuf = %s\n", sbuf);
    else {
        printf("dbuf = %s\n", dbuf);
        printf("return value is NULL\n");
    }
}
```

memcpy

レベル 2

書 式 `#include <string.h>`

```
void *memcpy(dest, orig, c, size) ;
void *dest; /* 転送先の領域を指すポインタ */
void *orig; /* 転送元の領域を指すポインタ */
char c; /* 転送を終了させる文字 */
size_t size; /* 転送バイト数 */
```

機 能 `orig` が指す領域から `dest` が指す領域へ文字 `c` が見つかるか、または転送したバイト数が `size` バイトに達するまで転送します。

戻 り 値 文字 `c` までを転送したときには、`dest` 内の `c` の次の位置を指すポインタを返します。
`orig` 内に文字 `c` と同じ文字がなく、`size` バイト転送したときは `NULL` を返します。

参 照 関 数 `memchr`, `memcmp`, `memcpy`, `memset`

プログラム例

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

#define SIZE 256

static char sbuf[SIZE] = "hello world.good bye";
static char dbuf[SIZE] = "";

main()
{
    char *p;
    int c;

    c = '.';

    if ((p = memcpy(dbuf, sbuf, c, SIZE)) != NULL)
        printf("dbuf = '%s'\n", dbuf);
    else {
        printf("dbuf = '%s'\n", dbuf);
        printf("return value is NULL");
    }
}
```

memchr

レベル 2

書 式 #include <string.h>

```
void *memchr(buffer, c, size) ;
void *buffer; /* 検索を行う領域を指すポインタ */
int c; /* 検索する文字 */
size_t size; /* 検索を行うバイト数 */
```

機 能 buffer が指す領域から size バイトの範囲で文字 c を検索します。

戻 り 値 buffer 内の c の位置を指すポインタを返します。
文字 c がなかった場合は、NULL を返します。

参 照 関 数 memccpy, memicmp, memcpy, memset

プログラム例

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

#define SIZE 16

static char buf[SIZE + 1] = "0123456789abcdef";

main()
{
    char *p;
    int c;

    for (c = '0'; c < 0x7f; c++) {
        printf("char = %c, ", c);
        if ((p = memchr(buf, c, SIZE)) == NULL)
            printf("not found\n");
        else
            printf("offset = %d\n", p - buf);
    }
}
```

memcmp

レベル 2

書 式 #include <string.h>

```
int memcmp(mem1, mem2, size) ;  
void *mem1; /* 比較領域 1 を指すポインタ */  
void *mem2; /* 比較領域 2 を指すポインタ */  
size_t size; /* 比較バイト数 */
```

機 能 mem1 が指す領域と mem2 が指す領域を size バイトの範囲だけ比較します。

戻 り 値 2 つの領域の内容の大小関係を表す下記の値を返します。

戻り値	内 容
負	mem1 は mem2 より小さい
0	mem1 と mem2 は等しい
正	mem1 は mem2 より大きい

参 照 関 数 memcmppy, memchr, memcpy, memcmapi, memicmp, memset, strcmp

プログラム例

```
#include <stdio.h>  
#include <string.h>  
  
#define SIZE 16  
  
static char mem1[SIZE + 1] = "0123456789abcdef";  
static char mem2[SIZE + 1] = "0123456789ABCDEF";  
  
main()  
{  
    int r;  
  
    if ((r = memcmp(mem1, mem2, SIZE)) == 0)  
        printf("mem1 == mem2\n");  
    else if (r < 0)  
        printf("mem1 < mem2\n");  
    else if (r > 0)  
        printf("mem1 > mem2\n");  
}
```


memcmp, memicmp

レベル 2

書 式 #include <string.h>

```
int memicmp(mem1, mem2, size);
int memcmp(mem1, mem2, size);

void *mem1; /* 比較領域 1 を指すポインタ */
void *mem2; /* 比較領域 2 を指すポインタ */
size_t size; /* 比較バイト数 */
```

機 能 mem1 が指す領域と mem2 が指す領域を size バイトの範囲だけ、大文字/小文字を区別しないで比較します。

memcmp, memicmp は、大文字/小文字を区別しない以外は memcmp 関数と同じです。

また、memcmp, memicmp は、関数名が異なるだけで、機能は同じです。

戻 り 値 2 つの関係を表す下記の値を返します。

戻り値	内 容
負	mem1 は mem2 より小さい
0	mem1 と mem2 は等しい
正	mem1 は mem2 より大きい

参 照 関 数 memcmp, strcmpi

プログラム例

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

#define SIZE   16

static char   mem1[SIZE + 1] = "0123456789abcdef";
static char   mem2[SIZE + 1] = "0123456789ABCDEF";

main()
{
    int        r;

    if ((r = memcmp(mem1, mem2, SIZE)) == 0)
        printf("mem1 == mem2\n");
    else if (r < 0)
        printf("mem1 < mem2\n");
    else if (r > 0)
        printf("mem1 > mem2\n");
}
```

例 10-1

memcpy, memmove, movedata, movmem

レベル 2

書 式 #include <string.h>

```
void *memcpy(dest, source, size) ;
void *memmove(dest, source, size) ;
void movedata(source, dest, size) ;
void movmem(source, dest, size) ;
```

```
void *source; /* 転送元領域のポインタ */
void *dest; /* 転送先領域のポインタ */
size_t size; /* 転送バイト数 */
```

機 能

source が指す領域の内容を、dest が指す領域に size バイト転送します。

これらの領域は、重なり合っている場合でも、正しく転送が行われます。

ただし、XC 以外の処理系では、memcpy は領域を重ねると正しく転送されない処理になっている可能性があるため、移植を行う場合には注意してください。

また、movedata は、他の処理系では引数の数が異なる場合がありますので、これも移植時には注意してください。

source と dest の指定順が異なるので、これらの関数を使用するときには注意してください。

またこれらの関数は、strcpy 関数などとは異なり、ヌル文字'¥0'には関係なく size バイトを転送します。

戻 り 値

memcpy, memmove は転送先領域へのポインタを返します。

movedata, movmem には戻り値はありません。

参 照 関 数

memccpy, memchr, memcmp, memset

プログラム例

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

static char des[] = "SHARP X68000";
static char src1[] = "Sharp";
static char src2[] = "sharp";

main()
{
    printf("des = %s\n", des);
    (void)memcpy(des, src1, strlen(src1));
    printf("des = %s\n", des);
    movmem(src2, des, strlen(src2));
    printf("des = %s\n", des);
}
```

左 右

左 右

左 右

左 右

memset, setmem

レベル 2

書 式 #include <string.h>

```
void *memset(mem, c, size) ;
void setmem(mem, size, c) ;
void *mem; /* 文字を埋める領域を指すポインタ */
int c; /* 設定する文字 */
size_t size; /* 設定バイト数 */
```

機 能 mem が指す領域の size バイトを文字 c で埋めます。

戻 り 値 memset は文字を埋めた領域を指すポインタを返します。
setmem には、戻り値はありません。

参 照 関 数 memccpy, memchr, memcmp, memcpy

プログラム例

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

#define SIZE 256

static char buf[SIZE];

main()
{
    void test(char *);

    (void)memset(buf, 'x', SIZE);
    test(buf);
    /* setmem */
    setmem(buf, SIZE / 2, 'o');
    test(buf);
}

static void test(char *p)
{
    int i, j;

    for (i = 0; i < 16; i++) {
        for (j = 0; j < 16; j++) {
            putchar(*p);
            putchar(' ');
            ++p;
        }
        putchar('\n');
    }
    putchar('\n');
}
```

m_free

レベル 3

書 式 #include <music.h>

```
int m_free(track) ;  
char track; /*  トラック番号   */
```

機 能 FM 音源および MIDI で使用する、track で指定されたトラックの残り容量を調べます。

戻 り 値 正常終了の場合は、残り容量をバイト数で返します。
FM 音源が使用不可の場合、-1 を返します。
また、楽譜データの最小単位は 3 バイトですので、残り容量が 2 バイト以下の場合は、それ以上トラックに書き込むことはできません。

min

レベル 2

書 式 `#include <stdlib.h>`

`(type)min(a, b);`

`(type)a;`

`(type)b;`

機 能

min は、a と b を比較し、値の小さい方を返すマクロです。

2つの引数と、戻り値の型は同じ型で、比較演算が可能な型でなければなりません。

引数には、副作用を引き起こすものや、関数などを指定すると、期待した結果と異なる場合がありますので注意してください。

戻 り 値

2つの引数のうち、小さい方の値を返します。

参 照 関 数

max

プログラム例

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

#define SIZE   10

static int    a[SIZE] = {3, 1, 4, 1, 5, 9, 2, 5, 7, 1};

main()
{
    int        m;
    int        i;

    m = a[0];
    for (i = 0; i < SIZE; i++)
        m = min(m, a[i]);
    printf("最小の値は%d\n", m);
}
```

m_init

レベル 3

書 式 #include <music2.h>

int m_init();

機 能 FM 音源（トラックの楽譜データ、音色）を初期化します。
m_assign で割りあてたトラックをデフォルトに戻します。

戻 り 値 正常終了の場合は 0 を返します。
FM 音源が使用不可の場合、-1 を返します。

```
#include <stdio.h>
#include <music2.h>

#define SIZE 10

static int a[SIZE] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

main()
{
    int i;
    int j;

    m = a[0];
    for (i = 0; i < SIZE; i++)
        m = min(a[i], m);
    printf("最小の値は%d\n", m);
}
```


mkdir

レベル 2

書 式 `#include <direct.h>`

```
int mkdir(pathname) ;
char *pathname; /* 新ディレクトリ名へのポインタ */
```

機 能 `pathname` で指定した、新しいディレクトリを作成します。

戻 り 値 正常終了ならば、正の数 (0 も) を返します。
 エラーならば -1 を返します。
 また、エラーが発生した場合、`errno` と `_doserrno` にそれぞれ c のエラー番号と OS のエラー番号をセットします。

参 照 関 数 `chdir`, `rmdir`

プログラム例

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

#define DIR1 "¥¥tmp2"
#define DIR2 "¥¥tmp2¥¥sub"

main()
{
    if (mkdir(DIR1) >= 0) {
        printf("%s is created.\n", DIR1);
        if (mkdir(DIR2) == 0) {
            printf("%s is created.\n", DIR2);
            (void)rmdir(DIR2);
            printf("%s is removed.\n", DIR2);
        }
        (void)rmdir(DIR1);
        printf("%s is removed.\n", DIR1);
    }
}
```

mktemp

レベル 2

書 式 `#include <io.h>`

```
char *mktemp(temp);  
char *temp;
```

機 能

引数で指定された文字列をもとに、カレントディレクトリに存在しないファイル名を作成します。

引数の temp が指す文字列は、次のように設定します。

```
char temp[ ] = "strXXXXXX";
```

"str"は新たに作成するファイル名の先頭部分となり、"XXXXXX"の部分の別の文字列に置き換えます。

なお、"XXXXXX"の部分は、'X'という文字（アスキーコード 0x58）で6文字以上なければならず、別の文字が混ざっていたり6文字に足りないときはエラーとなります。

"XXXXXX"の部分は、後ろから5文字分を、その時の時刻の分と秒をもとに算出した値を文字列に変換して置き換えます。

後ろから6文字目は、通常'0'ですが、このようにして新たに作成したファイル名がすでにカレントディレクトリに存在する場合は、次に示す規則に従って置き換えていきます。

```
'0'→'a'→'b'→'c'→……→'z'
```

したがって、例えば"tmp000044"という文字列が生成され、同じ名前のファイルがすでにカレントディレクトリに存在している場合には、mktemp は次に"tmpa00044"というファイルの存在の有無を調べ、もし無ければ、その文字列を返します。

以下に、temp に設定した文字列の変換の例を示します。

- "sampleXXXXXX" → "sample000300"
- "tmpXXXXXXX" → "tmpX010100"
- "TPXXXXXX" → "TPa01234"

("TP001234"が存在していた場合)

なお、mktemp は引数の temp が指す文字列を直接変更するので、mktemp の呼び出しの前に temp に設定されていた文字列は失われます。

したがって、mktemp の引数には“で囲まれた文字列を指定してはいけません。

戻り値

新たにファイル名が作成できた場合には、そのファイル名の文字列を指し示すポインタを返します。

引数が不正か、新たなファイル名を作成できない場合は、エラーとなり、NULL を返します。

参照関数

fopen, getpid, open

プログラム例

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

#define MAX 6
#define NAMLEN 16
#define TEMPLATE "pcsXXXXXX"

static char tnames[MAX][NAMLEN];

main()
{
    char *make_uniq(char *);
    int touch(char *);
    int i, j;

    for (i = 0; i < MAX; i++) {
        if (make_uniq(tnames[i]) == NULL) {
            puts("create error");
            break;
        }
        else if (touch(tnames[i]) != 0) {
            puts("touch error");
            break;
        }
        printf("%s is created.\n", tnames[i]);
    }
    for (j = 0; j < i; j++) {
        unlink(tnames[j]);
        printf("%s is removed.\n", tnames[j]);
    }
}

static char *make_uniq(char *name)
{
    char temp[NAMLEN];
    char *r;

    strcpy(temp, TEMPLATE);
    if ((r = mktemp(temp)) == NULL)
        return (NULL);
    strcpy(name, r);
    return (name);
}

static int touch(char *name)
{
    FILE *fp;

    if ((fp = fopen(name, "w")) == NULL)
        return (-1);
    (void)fclose(fp);
    return (0);
}
```

mktime

レベル 2

書 式 #include <time.h>

```
time_t mktime(time_ptr);
struct tm * time_ptr; /* tm 構造体に分解された時間情報 */
```

機 能 time_ptr が指す構造体に格納されているローカル時間を、time 関数が返す値と同じ形式のカレンダ時間に変換します。

また、tm 構造体のメンバ tm_wday, tm_yday は、変換する時には参照されませんが、変換後には正しい値が設定されます。

他のメンバも、正しい範囲に調整されます。

戻 り 値 変換に成功するとカレンダ時間を返します。

失敗すると、(-1)を time_t 型に変換した、すなわち(time_t)(-1)とキャストされた値を返します。

参 照 関 数 time, gmtime, localtime

プログラム例

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

main()
{
    struct tm dt;
    time_t time1;
    time_t time2;
    double diff;

    dt.tm_year = 2001 - 1900;
    dt.tm_mon = 1;
    dt.tm_mday = 1;
    dt.tm_hour = 0;
    dt.tm_min = 0;
    dt.tm_sec = 0;
    dt.tm_isdst = 0;
    time1 = mktime(&dt);

    time(&time2);

    diff = difftime(time1, time2);
    printf("21世紀まであと%g秒\n", diff);
}
```

modf

レベル 2

書 式 #include <math.h>

```
double modf(x, intptr) ;  
* double x; /* 浮動小数点値 */  
double *intptr; /* 整数部を格納する領域へのポインタ */
```

機 能 浮動小数点値 x を小数部と整数部に分けます。

戻 り 値 x と同じ符号の小数部を返し、x と同じ符号が付いた整数部は intptr に格納されます。

参 照 関 数 frexp, ldexp

プログラム例

```
#include <stdio.h>  
#include <math.h>  
  
main()  
{  
    double x;  
    double ix, fx;  
    int i;  
  
    for (i = -15; i <= 15; i++) {  
        x = (double)i / 10.0;  
        fx = modf(x, &ix);  
        printf("%+4.1f ->modf()-> %g, %g\n", x, ix, fx);  
    }  
}
```

mouse

レベル 3

書 式 #include <mouse.h>

```
int mouse(i) ;
int i; /* 機能コード */
```

機 能 マウスの制御を行います。
i に指定できる機能コードは次の通りです。

- 0 : マウスの初期化、ソフトキーボードの表示を許す
- 1 : マウスカーソルの表示
- 2 : マウスカーソルを消す
- 3 : マウスカーソルの表示状態の通知
- 4 : ソフトキーボードの表示をさせない

戻 り 値 i に 3 を指定した場合は、以下に示すマウスカーソルの表示状態を返します。

- 1 : 表示している
- 0 : 消えている

i に 0、1、2、4 を指定した場合は、0 を返します。

i に 0~4 以外を指定した場合は、エラーとして -1 を返します。

プログラム例

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

#define M_INIT      0
#define M_ON        1
#define M_OFF       2
#define M_STAT      3

main(int argc, char *argv[])
{
    if (argc == 1)
        puts(mouse(M_STAT) ? "on" : "off");
    else if (argc == 2) {
        if (strcmp(argv[1], "on") == 0)
            mouse(M_ON);
        else if (strcmp(argv[1], "off") == 0)
            mouse(M_OFF);
        else
            puts("Usage: mouse [on|off]");
    }
    else
        puts("Usage: mouse [on|off]");
}
```

m_play

レベル 3

書 式 #include <music.h>

```
int m_play(c1, c2, c3, c4, c5, c6, c7, c8) ;
int c1, c2, c3, c4, c5, c6, c7, c8; /* チャンネル */
```

機 能

FM 音源の指定したチャンネルに割りあてたトラックデータを演奏開始します。
指定するチャンネルの個数が7個以下の場合、余りの引数に 'NASI' を指定しなければなりません。
全チャンネルが 'NASI' ならば、1~8 チャンネルすべてを演奏開始します。
チャンネルは、1~8 が指定できます。

戻 り 値

正常終了の場合は 0 を返します。
引数のエラーなどの場合 0 以外の正の数を返します。
FM 音源が使用不可の場合、-1 を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

例プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <music.h>

#define M_INIT 0
#define M_ON 1
#define M_OFF 2
#define M_STAT 3

main(argc, argv, envp)
{
    if (argc == 1)
        puts("usage: m_play [STAT] ? [on] : [off]")
    else if (argc == 2)
        if (strcmp(argv[1], "on") == 0)
            m_play(M_ON);
        else if (strcmp(argv[1], "off") == 0)
            m_play(M_OFF);
        else
            puts("usage: m_play [on|off]");
    else
        puts("usage: m_play [on|off]");
}
```


プログラム例

```

#include      <string.h>
#include      <basic0.h>
#include      <music.h>

#define NASI  'NASI'

static char  as[256 + 1];

main()
{
    int      i;

    b_init();
    m_init();
    for (i = 1; i <= 6; i++) {
        m_alloc(i, 2000);
        m_assign(i, i);
    }
    m_trk(1, "Q7 @34 V9   O4 T66"); /* trumpet */
    m_trk(2, "Q7 @35 V10  O4 T66"); /* horn */
    m_trk(3, "Q7 @37 V10  O3 T66"); /* tuba */
    m_trk(4, "Q8 @6  V8   O5 T66"); /* cembalo */
    m_trk(5, "Q8 @6  V8   O5 T66"); /* cembalo */
    m_trk(6, "Q8 @6  V8   O4 T66"); /* cembalo */

    /* Part 1 */
    strcpy(as, "L16 B-4&B-B-AGF8G8DE-F8 : ");
    strcat(as, "<R8F8&FE-DC>B-8<DA>AB-<C8");
    strcat(as, ">F<FD>B-<E-8G8&GE-C>A&A8<F8 : ");
    strcat(as, "D4E4F8>B-4A8");
    m_trk(1, as);
    m_trk(4, as);

    /* Part 2 */
    strcpy(as, "L16R1 : ");
    strcat(as, ">B-FB-<DC8E-8&E-8DE {FEFEFEFE}8 &ED32E");
    strcat(as, "F4G8B-8E-8G8<C8C8& : ");
    strcat(as, ">C>FB-<D>G8B-8A8DE-FGE-F");
    m_trk(2, as);
    m_trk(5, as);

    /* Part 3 */
    strcpy(as, "L8 B-<DC>B-AGFE- : ");
    strcat(as, "D4F4G4<C4:");
    strcat(as, "D4&L16DD>B-G<C8E-8&E-C>AF : ");
    strcat(as, "B-4&B-B-AGF8G8DE-F8");
    m_trk(3, as);
    m_trk(6, as);

    m_play(NASI, NASI, NASI, NASI, NASI, NASI, NASI, N.
    b_exit(0);
}

```

msarea

レベル 3

書 式 #include <mouse.h>

```
int msarea (x1, y1, x2, y2) ;  
int x1; /* マウ斯卡ーソルの移動範囲の始点 X 座標 */  
int y1; /* マウ斯卡ーソルの移動範囲の始点 Y 座標 */  
int x2; /* マウ斯卡ーソルの移動範囲の終点 X 座標 */  
int y2; /* マウ斯卡ーソルの移動範囲の終点 Y 座標 */
```

機 能 マウ斯卡ーソルの移動範囲を設定します。

x1 にはマウ斯卡ーソルの移動範囲の始点 X 座標を指定します。

y1 にはマウ斯卡ーソルの移動範囲の始点 Y 座標を指定します。

x2 にはマウ斯卡ーソルの移動範囲の終点 X 座標を指定します。

y2 にはマウ斯卡ーソルの移動範囲の終点 Y 座標を指定します。

かならず始点が左上、終点が右下でなければなりません。

また、それぞれの指定可能範囲は以下の通りです。

$0 \leq x1 \leq 1022$

$0 \leq y1 \leq 1006$

$x1 < x2 \leq 1023$

$y1 < y2 \leq 1007$

戻 り 値 正常終了のとき、0 を返します。

エラーのとき、-1 を返します。

プログラム例

```

#include      <stdio.h>
#include      <conio.h>
#include      <basic0.h>
#include      <mouse.h>
#include      <graph.h>

main()
{
    b_init();
    screen(2, 0, 1, 1);
    vpage(1);
    mouse(0);
    msarea(0, 0, 500, 400);
    box(0, 0, 500, 400, 15, 0xffff);
    mouse(1);
    puts("move mousr cursor! (0, 0) <---> (500, 400)");
    while (kbhit() == 0)
    {
        b_exit(0);
    }
}

```

書 式 #include <mouse.h>

```
int msbth(n, b, t);
```

```
int n; /* マウスボタンの状態の指定 */
```

```
int b; /* マウスボタンの指定 */
```

```
int t; /* 待ち時間 */
```

機 能

マウスボタンが、指定した状態になるまでの時間を通知します。

n にはマウスボタンの状態を指定します。

0 を指定するとマウスボタンを離すまで、1 を指定するとマウスボタンを押すまでの時間が測定されます。

b にはマウスボタンの左右の指定を行います。

0 で左ボタン、1 で右ボタンとなります。

t には待ち時間の最大値を指定します。

0～65535 までを指定します。

0 を指定すると、指定された状態が発生するまで待ち続けます。

戻 り 値

待ち時間を返します (1～65534)。

ドラッグが行われた場合は 0、待ち時間を超えた場合は -1 を返します。

引数のエラーの場合は -2 を返します。

プログラム例

```

#include <stdio.h>
#include <basic0.h>
#include <mouse.h>
#include <graph.h>

main()
{
    int d_clk(void);
    int i, j;
    int ni, nj;

    b_init();
    screen(2, 0, 1, 1);
    vpage(1);
    mouse(0);
    msarea(0, 0, 767, 511);
    box(0, 0, 767, 511, 15, 0xffff);
    mouse(1);
    puts("move line by double click!");
    for (i = j = 0; d_clk() != 1; i = ni, j = nj) {
        line(i, 0, i, 511, 0, 0xffff);
        line(0, j, 767, j, 0, 0xffff);
        box(0, 0, 767, 511, 15, 0xffff);
        mspos(&ni, &nj);
        line(ni, 0, ni, 511, 12, 0xffff);
        line(0, nj, 767, nj, 12, 0xffff);
    }
    b_exit(0);
}

static int d_clk(void)
{
    int x, y, bl, br;
    int f;

    msstat(&x, &y, &bl, &br);
    if (bl != -1)
        return (0);
    f = msbtn(0, 0, 80);
    if (10 > f || f > 70)
        return (0);
    f = msbtn(1, 0, 80);
    if (10 > f || f > 70)
        return (0);
    f = msbtn(0, 0, 80);
    if (10 > f || f > 70)
        return (0);
    return (1);
}

```

mpos

レベル 3

書 式 #include <mouse.h>

```
int mpos(x, y);  
int *x; /* X座標を格納する変数を指すポインタ */  
int *y; /* Y座標を格納する変数を指すポインタ */
```

機 能 マウスカーソルの座標を変数に代入します。
x にはマウスカーソルの X 座標を代入する変数を指すポインタを指定します。
y には Y 座標を代入する変数を指すポインタを指定します。

戻 り 値 上位 16 ビットに X 座標、下位 16 ビットに Y 座標が返ります。

プログラム例

```
#include <stdio.h>  
#include <conio.h>  
#include <basic0.h>  
#include <mouse.h>  
#include <graph.h>  
  
main()  
{  
    int x, y;  
  
    b_init();  
    screen(2, 0, 1, 1);  
    vpage(1);  
    mouse(0);  
    msarea(0, 0, 767, 511);  
    box(0, 0, 767, 511, 15, 0xffff);  
    mouse(1);  
    locate(10, 10);  
    puts("move mouse cursor!(0, 0) <--->(767, 511)");  
    while (kbhit() == 0) {  
        mpos(&x, &y);  
        locate(10, 11);  
        printf("mouse cursor (X) = %3d\n", x);  
        locate(10, 12);  
        printf("mouse cursor (Y) = %3d\n", y);  
    }  
    b_exit(0);  
}
```

msstat

レベル 3

書 式 #include <mouse.h>

```
int msstat(x, y, bl, br) ;
int  *x; /* X方向の相対移動量を格納する変数を指すポインタ */
int  *y; /* Y方向の相対移動量を格納する変数を指すポインタ */
int  *bl; /* 左ボタンの状態を格納する変数を指すポインタ */
int  *br; /* 右ボタンの状態を格納する変数を指すポインタ */
```

機 能 マウスの状態を変数に代入します。

x には X 方向の相対移動量を代入する変数を指すポインタを指定します。

−128〜127 が代入されます。

y には Y 方向の相対移動量を代入する変数を指すポインタを指定します。

−128〜127 が代入されます。

bl には左ボタンの状態を代入する変数を指すポインタを指定します。

左ボタンを押していた場合には−1 が、押していない場合には0 が代入されます。

br には右ボタンの状態を代入する変数を指すポインタを指定します。

右ボタンを押していた場合には−1 が、押していない場合には0 が代入されます。

戻 り 値

上位から8ビットごとに、X の移動量、Y の移動量、左ボタンの状態、右ボタンの状態が返ります。

プログラム例

```
#include <conio.h>
#include <basic0.h>
#include <mouse.h>
#include <graph.h>

main()
{
    int x, y;
    int bl, br;

    b_init();
    screen(2, 0, 1, 1);
    vpage(1);
    mouse(0);
    msarea(0, 0, 767, 511);
    box(0, 0, 767, 511, 15, 0xffff);
    mouse(1);
    locate(10, 10);
    puts("move mouse cursor!(0, 0) <--->(767, 511)");
    locate(10, 11);
    cputs("mouse cursor relative movement");
    locate(10, 14);
    cputs("mouse button status");
    while (kbhit() == 0) {
        msstat(&x, &y, &bl, &br);
        locate(18, 12);
        cprintf(" (X) = %4d", x);
        locate(18, 13);
        cprintf(" (Y) = %4d", y);
        locate(18, 15);
        cprintf(" (LEFT) = %4d", bl);
        locate(18, 16);
        cprintf(" (RIGHT) = %4d", br);
    }
    locate(0, 17);
    b_exit(0);
}
```


m_stat

レベル 3

書 式 #include <music.h>

```
int m_stat(channel) ;
int channel ; /* チャンネル番号 */
```

機 能 FM 音源または MIDI の channel で指定したチャンネルの演奏状態を調べます。
channel には 1~24 を指定します。
0 または 'NASI' を指定すると、1~24 すべてのチャンネルの状態を調べます。

戻 り 値 正常終了の場合はチャンネルの状態を返します。
このとき、channel に指定した値により以下のようになります。

● channel が 1~8 のとき (本体の FM 音源)

0 : 演奏中でない
1 : 演奏中

● channel が 9~24 のとき (MIDI ボード出力)

0 : 演奏中でない
1 : 演奏中

ただし、演奏中であっても md_off によって出力が停止している場合があります。

また、MIDI ボードがない場合には、常に 0 になります。

● channel が 0 または 'NASI' のとき

ビット 0~7 : 1~8 チャンネルの状態
ビット 9~23 : 9~24 チャンネルの状態
(MIDI ボードがないとき、ビットは 0 になります)
ビット 24~31 : 常に 0

ビットが 0 で演奏中でない、1 で演奏中であることを表します。

FM 音源が使用不可の場合、-1 を返します。

m_stop

レベル 3

書 式 #include <music.h>

```
int m_stop(c1, c2, c3, c4, c5, c6, c7, c8) ;  
int c1, c2, c3, c4, c5, c6, c7, c8; /* チャンネル */
```

機 能 FM 音源のチャンネルを指定して演奏を一時中断します。
指定するチャンネルの個数が7個以下の場合、余りの引数に 'NASI' を指定しなければなりません。
全チャンネルが 'NASI' ならば、1~8 チャンネル中、現在演奏していたチャンネルすべてを一時中断します。
チャンネルは、1~8 が指定できます。

戻 り 値 正常終了の場合は 0 を返します。
引数のエラーなどの場合、0 以外の正の数を返します。
FM 音源が使用不可の場合、-1 を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

m_tempo

レベル 3

書 式 #include <music.h>

```
int m_tempo(tempo) ;  
char tempt; /* テンポ */
```

機 能

FM 音源および MIDI の演奏時のテンポを指定します。

テンポは、1 分間に 4 分音符をいくつ演奏できるかで指定します。

たとえば、120 を指定すると、4 分音符 120 拍で 1 分になります。

テンポに指定できる範囲は、32~200 です。

割り込み処理や、ディスクアクセスなどによって、演奏速度に誤差が発生する場合があります。

戻 り 値

正常終了の場合は 0 を返します。

引数のエラーなどの場合、0 以外の正の数を返します。

FM 音源が使用不可の場合、-1 を返します。

なお、エラー番号とその意味は、m_alloc 関数を参照してください。

m_trk

レベル 3

書 式 #include <music.h>

```
int m_trk(t, st);  
char t; /*  トラック番号   */  
char *st; /*   MML データへのポインタ   */
```

機 能 MML による楽譜データを t で指定したトラックに書き込みます。
MML の書式については、本体添付の「BASIC マニュアル」を参照してください。

戻 り 値 正常終了の場合は 0 を返します。
引数のエラーや MML の書式誤りなどの場合、0 以外の正の数を返します。
FM 音源が使用不可の場合、-1 を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

m_vget

レベル 3

書 式 #include <music2. h>

```
int m_vget(vo, ca) ;
char vo; /* 音色番号 */
char *ca; /* 音色データを格納する配列へのポインタ */
```

機 能 FM音源のvoで指定した音色番号の音色データをcaが指す配列に読み込みます。
voには音色番号を指定します(1~200。ただし69~200は登録されていません)。
caには、読み込んだ音色データを格納するための二次元配列へのポインタを指定します。
音色データの形式は次の通りです。

```
char ca [5] [11] ;
```

	0	1~4 (オペレータ)
0	フィードバック/アルゴリズム (0~63)	AR (0~31)
1	スロットマスク (0~15)	D1R (0~31)
2	ウェーブフォーム (0~3)	D2R (0~31)
3	シンクロ (0, 1)	RR (0~15)
4	スピード (0~255)	D1L (0~15)
5	PMD (0~127)	TL (0~127)
6	AMD (0~127)	KS (0~3)
7	PMS (0~7)	MUL (0~15)
8	AMS (0~3)	DT1 (0~7)
9	L, R PAN (0~3)	DT2 (0~3)
10	未使用	AMS Enable (0, 1)

上記の表中、() 内の数値はそれぞれのデータの範囲を示します。

音色データの詳細については、本体添付の「BASIC マニュアル」を参照してください。

戻 り 値 正常終了の場合は0を返します。
引数のエラーなどの場合0以外の正の数を返します。
FM 音源が使用不可の場合、-1を返します。
なお、エラー番号とその意味は、m_alloc関数を参照してください。

書 式 #include <music2.h>

```
int m_vset(vo, ca) ;
char vo; /* 音色番号 */
char *ca; /* 音色データを格納してある配列へのポインタ */
```

機 能 配列に格納してある FM 音源用の音色データを、音色に登録します。
vo には音色番号を指定します (1~200。ただし 1~68 は登録済み)。
ca には音色データを格納してある二次元配列へのポインタを指定します。
音色データの形式は次の通りです。

```
char ca [5] [11] ;
```

	0	1~4 (オペレータ)
0	フィードバック/アルゴリズム (0~63)	AR (0~31)
1	スロットマスク (0~15)	D1R (0~31)
2	ウェーブフォーム (0~3)	D2R (0~31)
3	シンクロ (0, 1)	RR (0~15)
4	スピード (0~255)	D1L (0~15)
5	PMD (0~127)	TL (0~127)
6	AMD (0~127)	KS (0~3)
7	PMS (0~7)	MUL (0~15)
8	AMS (0~3)	DT1 (0~7)
9	L, RPAN (0~3)	DT2 (0~3)
10	未使用	AMS Enable (0, 1)

上記の表中、() 内の数値はそれぞれのデータの範囲を示します。

音色データの詳細については、本体添付の「BASIC マニュアル」を参照してください。

戻 り 値 正常終了の場合は 0 を返します。

引数のエラーなどの場合 0 以外の正の数を返します。

FM 音源が使用不可の場合、-1 を返します。

なお、エラー番号とその意味は、m_alloc 関数を参照してください。

nfiles

レベル 2

書 式 #include <io. h>

```
char *nfiles(buff, atr, datetime, len) ;
char *buff; /* 検索用バッファへのポインタ */
char *atr; /* 見つけたファイルの属性を格納する領域へのポインタ */
int *datetime; /* 見つけたファイルの作成日時を格納する領域へのポ
ンタ */
size_t *len; /* 見つけたファイルのバイト数を格納する領域へのポインタ */
```

機 能

ファイルの次の候補を検索し、見つけたファイルの情報を atr, datetime, len で指定した領域に格納し、戻り値として見つけたファイルのファイル名へのポインタを返します。

buff は、files 関数で指定した検索用バッファへのポインタです。

この関数を呼び出す前には、files 関数を呼び出しておかねばなりません。

atr は、見つけたファイルの属性を格納する領域へのポインタです。

atr の指す領域のサイズは 1 バイト以上必要です。

* atr に格納された属性は、各ビットが 1 の場合に以下の意味を表しています。

ビット 0 : 読み込み専用

ビット 1 : 隠しファイル

ビット 2 : システムファイル

ビット 3 : ボリューム名

ビット 4 : ディレクトリ

ビット 5 : 通常ファイル

datetime は、見つけたファイルの作成日時を格納する領域へのポインタです。

datetime の指す領域のサイズは 4 バイト以上必要です。

また、* datetime に格納される作成日時の形式は以下の通りです。

ビット 31~25 : 1980 年からの相対年数(0~99)

ビット 24~21 : 月(1~12)

ビット 20~16 : 日(1~31)

ビット 15~11 : 時(0~23)

ビット 10~5 : 分(0~59)

ビット 4~0 : 秒/2(0~29)

len は、見つけたファイルのバイト数を格納する領域へのポインタです。

len の指す領域のサイズは4バイト以上必要です。

files 関数を実行した後で、nfiles をエラーが返されるまで実行することで、files 関数で指定しておいた条件を満たすファイル全てを検索することができます。

戻り値

正常終了ならば、見つけたファイルのファイル名へのポインタを返します。

なお、このファイル名は buff で指定した検索用バッファに作成されるため、同一の検索用バッファを指定して files 関数や、nfiles を呼び出すと以前の内容は失われることに注意してください。

エラーならば NULL を返します。

参照関数

chgfa, chgft, files

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>

#define A_RDO 0x01 /* 読み込み専用 */
#define A_HDN 0x02 /* 隠しファイル */
#define A_SYS 0x04 /* システムファイル */
#define A_VOL 0x08 /* Volumeラベル */
#define A_DIR 0x10 /* ディレクトリ */
#define A_FIL 0x20 /* 通常ファイル */

static char *pattern = "%Yusr%Yinclude%Y*.h";
static char buf[150];
static char atr;
static long dt;
static size_t len;

main()
{
    char *name;

    name = files(buf, &atr, &dt, &len, pattern, A_FIL);
    if (name != NULL)
        printf("%s\n", name);
    else
        exit(1);

    while ((name = nfiles(buf, &atr, &dt, &len)) != NULL)
        printf("%s\n", name);

    exit(0);
}
```


offsetof

レベル 2

書 式 #include <stddef.h>

```
size_t offsetof (tag, rnrber) ;
(type)tag;
(identifier)member;
```

機 能

offsetof は、tag の構造体での構造体メンバ member の位置を求めるマクロです。

tag には、構造体名を指定します。

member には、メンバ名を指定します。

ビットフィールドの構造体は指定できません。

戻 り 値

tag の構造体での、構造体メンバ member の位置を返します。

書 式	機 能
O_APPEND	ファイルの末尾にデータを追加する。
O_CREAT	ファイルが存在しない場合は、指定されたモードで新規に作成する。
O_EXCL	O_CREAT とともに指定すると、ファイルが存在しない場合にのみ作成される。
O_RDONLY	読み出し専用としてファイルを開く。
O_RDWR	読み書きのためにファイルを開く。
O_TRUNC	ファイルの長さを 0 にリセットする。
O_WRONLY	書き出し専用としてファイルを開く。
O_BINARY	バイナリモードでファイルを開く。
O_TEXT	テキストモードでファイルを開く。

open

レベル 2

書 式

```
#include <fcntl.h>
```

```
#include <stat.h>
```

```
#include <io.h>
```

```
int open(pathname, omode [, pmode] );
```

```
char *pathname; /* ファイル名へのポインタ */
```

```
int omode; /* モード */
```

```
int pmode; /* アクセスモード */
```

機 能

pathname で指定したファイルを omode で指定したモードでオープンします。

omode は下記に示す定数の組み合わせで指定します。

複数個の定数を組み合わせるときは、OR ' | ' でつなげます。

例: O_TEXT | O_RDWR

omode の値	内 容
O_APPEND	ファイルポインタを書き込み操作の前にエンドオブファイルに移動する
O_CREAT	書き込みのために新しいファイルを作る ただし、ファイルが存在していればこのモードは無効となる
O_EXCL	ファイルが存在した場合にエラー値を返す O_CREAT とともに指定したときのみに有効となる
O_RDONLY	読み出し専用としてファイルをオープンする O_RDWR、O_WRONLY を同時に指定できない
O_RDWR	読み書きのためにファイルをオープンする O_RDONLY、O_WRONLY を同時に指定できない
O_TRUNC	ファイルが存在していれば、ファイルの内容を空にする O_RDWR または O_WRONLY とともに指定したときのみに有効となる
O_WRONLY	書き込み専用としてファイルをオープンする O_RDONLY、O_RDWR を同時に指定できない
O_BINARY	ファイルをバイナリモードでオープンする
O_TEXT	ファイルをテキストモードでオープンする

O_TRUNC は、既存のファイルの内容をすべて切り捨ててしまいます。
慎重に使用してください。

omode に O_CREAT を指定したときは、必ず pmode を指定します。
ただしファイルがすでに存在している場合は、pmode は無視されます。
pmode の値は chmod 関数での指定と同じです。

戻り値

オープンされたファイルのファイルハンドルを返します。
エラーが発生したときは -1 を返し、errno に下記の値がセットされます。

errno の値	内 容
EACCES	指定したファイル名がディレクトリである またはアクセスモードが違っている
EEXIST	O_CREAT、O_EXCL を指定したが、ファイルがすでに 存在している
EMFILE	オープンするファイルの数が多すぎる
ENOENT	指定したファイル名またはパス名が見つからない

参照関数

access, chmod, close, creat, dup, dup2, fopen, sopen, umask

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <io.h>
#include <stat.h>

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

    file = (argc > 1) ? argv[1] : "file";

    if ((fn = open(file, O_RDONLY)) == -1) {
        printf("%sを開けません。¥n", file);
        exit(1);
    }
    printf("%sを開しました。¥n", file);

    if (close(fn) == -1) {
        printf("%sをcloseできません。¥n", file);
        exit(1);
    }
    printf("%sをcloseしました。¥n", file);

    exit(0);
}
```

paint

レベル 3

書 式 #include <graph.h>

```
void paint(x, y, p);
int x; /* X座標 */
int y; /* Y座標 */
int p; /* パレットコード */
```

機 能 グラフィック画面の x、y で指定された単色領域を p で塗りつぶします。
 x には領域内の 1 点の X 座標を指定します (−32768~32767)。
 y には領域内の 1 点の Y 座標を指定します (−32768~32767)。
 p にはパレットコードを指定します (0~65535)。
 画面モードにより最大値が異なります。

戻 り 値 戻り値はありません。

参 照 関 数 screen

プログラム例

```
#include <stdlib.h>
#include <basic0.h>
#include <graph.h>

main()
{
    int i;
    int b;
    int c;

    b_init();
    screen(1, 3, 1, 1);
    vpage(1);
    for (i = 0; i < 500; i++) {
        b = rand() * 2 + 1;
        c = rand() * 2 + 1;
        circle(255, 255, 100, b, 0, 360, 256);
        paint(255, 255, c);
    }
    b_exit(0);
}
```

palet

レベル 3

書 式 #include <graph.h>

```
void palet(p, c) ;  
int p; /* パレットコード */  
int c; /* 色コード */
```

機 能 パレットに色コードを割りあてます。

p にはパレットコードを指定します (0~65535)。

画面モードにより最大値が異なります。

c には色コードを指定します (0~65535)。

なお、65536 色モードでは、1つの色を変化させると、同時に関連する 256 色が変化しますので、原則的に色コードとパレットコードは同じ値を指定します。

戻 り 値 戻り値はありません。

参 照 関 数 screen

プログラム例

```
#include <basic0.h>  
#include <graph.h>  
  
main()  
{  
    int a, b, c, d, e;  
    int f;  
  
    b_init();  
    screen(1, 3, 1, 1);  
    vpage(1);  
    for (a = 0; a < 2; a++) {  
        for (b = 31; b >= 0; b--) {  
            for (c = 31; c >= 0; c--) {  
                for (d = 0; d < 192; d++) {  
                    f = hsv(d, c, b) + 1 - a;  
                    palet(15, f);  
                    for (e = 0; e < 300; e++)  
                        ;  
                }  
            }  
        }  
    }  
    b_exit(0);  
}
```

perror

レベル 2

書 式 `#include <stdio.h>`

```
void perror(string) ;
char *string; /* ユーザー定義エラーメッセージへのポインタ */
```

機 能

標準エラー出力 (stderr) に対してエラーメッセージを出力します。

エラーメッセージは string、コロン (:), そしてエラーを起したライブラリに対するシステムエラーメッセージからなります。

string に空文字列か NULL を指定すると、システムエラーメッセージのみを出力します。エラー番号は、グローバル変数 `errno` が使用されます。

もし、`errno` が 0 以上の値ならば、`perror` は何もしません。

システムエラーメッセージは、`errno` を使ってエラー番号順に並んだ文字列のポインタ配列 `sys_errlist` から探し出されます。

変数 `sys_nerr` の値は、`sys_errlist` 配列の要素数です。

`errno` の値は何らかの関数の呼び出し時におけるエラーの発生によって変化します。

この関数で参照されるグローバル変数には、以下のものがあります。

```
int errno; /* エラー番号 */
int sys_nerr; /* sys_errlist の配列要素数 */
char *sys_errlist []; /* エラーメッセージのポインタ配列 */
```

戻 り 値 戻り値はありません。

参 照 関 数 `strerror`

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>

main(int argc, char *argv[])
{
    char *name;
    FILE *fp;

    name = (argc > 1) ? argv[1] : "file";
    if ((fp = fopen(name, "r")) == NULL) {
        perror(name);
        abort();
    }
    (void)fclose(fp);
    exit(0);
}
```

pi

レベル 3

書 式 #include <basic0.h>

double pi();

機 能 円周率 π の値を求めます。

戻 り 値 π の値を返します。

参 照 関 数 b_pi

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>

main(int argc, char *argv[])
{
    char *name;
    FILE *fp;

    name = (argc > 1 ? argv[1] : "file");
    if ((fp = fopen(name, "r")) == NULL)
        perror(name);
    printf("%s\n", name);

    (void)fclose(fp);
    exit(0);
}
```


point

レベル 3

書 式 `#include <graph.h>`

```
int point(x, y);
int x; /* X座標 */
int y; /* Y座標 */
```

機 能 グラフィック画面上の指定座標のドットのパレットコードを調べます。

x には調べる点の X 座標を指定します (−32768~32767)。

y には調べる点の Y 座標を指定します (−32768~32767)。

戻 り 値 指定座標のドットのパレットコードを返します。

ただし、window 関数によって設定されたクリッピング領域の範囲外の座標を指定した場合は、たとえ指定座標にドットがあったとしても、常に 0 を返します。

参 照 関 数 `pset, window`

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <basic0.h>
#include <graph.h>

main()
{
    int i;
    int b;
    int c;
    int d;

    b_init();
    screen(1, 3, 1, 1);
    vpage(1);
    for (i = 0; i < 500; i++) {
        b = rand() * 2 + 1;
        c = rand() * 2 + 1;
        circle(255, 255, 100, b, 0, 360, 256);
        paint(255, 255, c);
        d = point(255, 255);
        printf("palet code = %d\n", d);
    }
    b_exit(0);
}
```

書 式 #include <basic0.h>

int pos();

機 能 カーソルの現在位置の X 座標を得ます。

戻 り 値 カーソルの X 座標を返します。

参 照 関 数 csrlin

参 照 関 数

参 照 関 数

参 照 関 数

```

#include <stdio.h>
#include <stdlib.h>
#include <basic0.h>
#include <graph.h>

main()
{
    int i;
    int p;
    int c;
    int d;

    p_init();
    screen(1, 3, 1, 1);
    vpage(1);
    for (i = 0; i < 500; i++)
    {
        p = rand() % 2 * 1;
        c = rand() % 2 * 1;
        circle(325, 255, 160, p, 0, 360, 300);
        printf(325, 255, c);
        d = point(325, 255);
        printf("pixel code = %d\n", d);
    }
    _exit(0);
}

```

pow

レベル 2

書 式 `#include <math.h>`

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

機 能 x の y 乗を計算します。

戻 り 値 x^y の値を返します。
 x が 0 で y が 0 か負の数の場合、または x が負の数で y が整数でない場合の、
 戻り値は不定で、`errno` に `EDOM` がセットされます。

参 照 関 数 `exp`, `log`, `sqrt`

プログラム例

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

main()
{
    double x;
    double y;
    int i;
    int j;

    for (i = -32; i <= 32; i++) {
        for (j = -32; j <= 32; j++) {
            x = (double)i;
            y = (double)j;
            printf("pow(%.1f, %.1f) = %g\n",
                x, y, pow(x, y));
        }
    }
}
```

printf

レベル 2

書 式 #include <stdio.h>

```
int printf(format_string [, arg. . . ] ) ;
char *format_string; /* 書式文字列へのポインタ */
```

機 能 format_string で指定した書式に従って出力を変換し、標準出力 stdout に出力します。

format_string に続く引数は、書式に従って変換され出力されます。

format_string 中のフォーマット指定の出現順、およびそれが要求する型と、引数 arg の出現順およびその型は一致していなければなりません。

フォーマット指定の書式を以下に示します。

% [flag] [length] [. digit] [lentype] type

“%”と type 以外は省略できます。

フォーマット指定の各フィールドの内容は次の通りです。

フィールド	内 容
%	書式指定の始まりを示す 省略できない
flag	符号、空白、小数点、8 進、16 進などの出力の修飾を指定する
length	出力文字の最小文字数を指示する
digit	最小桁数、または小数点以下の桁数を指定する
lentype	引数の型を指定する
type	変換のタイプを指定する

“%”のうしろに format_string として無意味な文字があると、そのまま出力します。

各フィールドの内容は次の通りです。

[flag]

flag の値と内容は次の通りです。

flag	内 容
—	length の桁数内で左揃えて出力する 省略すると右揃えて出力する
+	出力数値が正の数のときは“+”を、負の数のときは“−”をつけて出力する 省略すると、負の数のときのみ“−”をつけ、正の数のときは符号をつけない
空白	出力数値が正の数のときは1文字分の空白を、負の数のときは“−”をつける 省略すると、負の数のときのみ“−”をつけ、正の数のときは空白をつけない flag に“+”が指定されているときは、この指定は無効になる
#	type の値が、0、x、X、b、B のとき、出力数値の先頭に0、0x、0X、0b、0B をつける 省略すると何もつけない
	type の値がf、e、E のとき、小数点以下に数字がなくても小数点を付ける 省略すると、小数点以下に数字があるときのみ小数点を付ける
	type の値がg、G のとき、小数点以下の数字がdigit で指定した桁数に足りなくても、切り捨てや余分な0の削除は行わない 省略すると、小数点以下に数字があるときのみ小数点以下か付いて、余分な0は削除する
	type の値がc、d、u、s のときは無視される

【length】

出力する最小文字数を10進数の整数で指定します。

出力する文字数がlengthよりも少ない場合は、文字の左側を空白で埋めて右揃え（マイナスフラグが指定されているときは逆）で出力されます。

出力する文字数がlengthよりも多い場合、またはlengthが指定されていない場合は、すべての文字を出力します。

length を引数で指定する場合には、アスタリスク (*) を使用します。
length に対応する引数は、フォーマット出力する引数の前になければならず、int 型でなければなりません。

【digit】

digit は出力する最大文字数または小数点以下の桁数を正の 10 進数の整数で指定します。

digit を引数で指定する場合には、アスタリスク (*) を使用します。

digit に対応する引数は、フォーマット出力する引数の前になければならず、int 型でなければなりません。

digit の指定方法、および省略した場合のデフォルト値は次の通りです。

type の値	内 容
d u b B o x X	出力する最小桁を指定する 桁数が digit より少ない場合は 0 を付加する 桁数が digit を超えても値の切り捨ては行われず、そのまま出力する 省略したとき、0 を指定されたときは“1”とみなす
f e E	小数点以下の桁数を指定する 出力する最後の桁は四捨五入される 省略したときは 6 桁になる 0 を指定したとき、またはピリオドのみが記述されたときは、小数点以下の数字を切り捨てる
g G	出力する有効値の最大桁数を指定する 省略したときは有効値のすべてを出力する
c	無効
s	出力する最大文字数を指定する 省略したときは、文字列の末尾のヌル文字に達するまで出力する

【lentype】

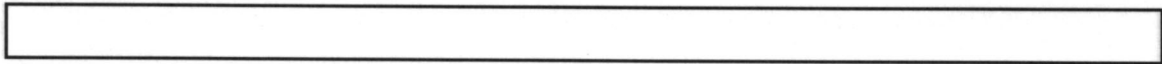
lentype は出力する引数の型を指定します。

lentype の内容は次の通りです。

lentype の値	内 容
l L	type の値が d、u、b、o、x のとき、引数が long か unsigned long であることを指定する
h H	type の値が d、u、b、o、x のとき、引数が short か unsigned short であることを指定する

【type】

type の値	引 数	内 容
d i	整数型	符号つき 10 進整数
u	整数型	符号なしの 10 進整数
b	整数型	符号なしの 2 進整数
B	整数型	符号なしの 2 進整数
o	整数型	符号なしの 8 進整数
x	整数型	符号なしの“abcdef”を使った 16 進整数
X	整数型	符号なしの“ABCDEF”を使った 16 進整数
f	double 型	double 値を [−] ddd. ddd の小数点形式に変換して出力する 小数点以下の桁数は digit により指定されます
e	double 型	double 値を [−] d.ddd e [−] ddd の指数形式に変換して出力する 小数点以下の桁数は digit により指定される
E	double 型	指数の前に“e”の代わりに“E”が使われる以外は e フォーマットと同じ



type の値	引 数	内 容
g	double 型	double 値を、f フォーマット（小数点形式）または e フォーマット（指数形式）のうち、結果が短くなる形式に変換して出力する 指数部が-4 以下または桁数が digit よりも大きくなる場合は e フォーマットになる
G	double 型	指数の前に“e”の代わりに“E”が使われる以外は g フォーマットと同じ
c	文 字	1 文字
s	文字列	スル文字（‘\0’）で終了する文字列 または digit の数の文字列
p	ポインタ型	ポインタのアドレスを 16 進数出力
n	int * 型	現在までに出力した文字数を引数に代入する 出力は行なわない

戻 り 値 出力した文字数を返します。

参 照 関 数 fprintf, scanf, sprintf

プログラム例

```
#include <stdio.h>

main()
{
    printf("hello world\n");
    printf("integer[%d]\n", 200);
    printf("integer[%x]\n", 200);
    printf("float[%f]\n", 3.1415926);
    printf("float[%g]\n", 3.1415926);
    printf("character[%c]\n", 'A');
    printf("string[%s]\n", "string");
}
```


pset

レベル 3

書 式 #include <graph.h>

```
void pset(x, y, p);
int x; /* X座標 */
int y; /* Y座標 */
int p; /* パレットコード */
```

機 能

グラフィック画面に点（ドット）を描画します。

x には点を描く X 座標を指定します（-32768～32767）。

y には点を描く Y 座標を指定します（-32768～32767）。

p にはパレットコードを指定します（0～65535）。

画面モードにより最大値が異なります。

戻 り 値

戻り値はありません。

参 照 関 数

screen, point

プログラム例

```
#include <basic0.h>
#include <stdlib.h>
#include <graph.h>

main()
{
    int i;
    int x;
    int y;
    int c;

    b_init();
    screen(2, 0, 1, 1);
    vpage(1);
    for (i = 0; i < 5000; i++) {
        x = rand() % 768;
        y = rand() % 200;
        c = rand() % 16;
        pset(x, y, c);
    }
    b_exit(0);
}
```

put

レベル 3

書 式 #include <graph.h>

```
void put(x1, y1, x2, y2, buf, size) ;
int x1; /* 始点 X 座標 */
int y1; /* 始点 Y 座標 */
int x2; /* 終点 X 座標 */
int y2; /* 終点 Y 座標 */
void *buf; /* 画像データ格納配列へのポインタ */
int size; /* 画像データのバイト数 */
```

機 能

buf が示す配列に格納された size バイトの画像データを、グラフィック画面の長方形の領域に描画します。

x1 には始点 X 座標を指定します。

y1 には始点 Y 座標を指定します。

x2 には終点 X 座標を指定します。

y2 には終点 Y 座標を指定します。

始点と終点は window で指定したクリッピング座標の中でなければいけません。

buf には、描画する画像データが格納してある領域へのポインタを指定します。

この領域には、size バイト分画像データが格納されていなければなりません。

siz には、画像データのバイト数を指定します。

画面モードにより、以下のバイト数を指定します。

16 色モード : (総ドット数+1) ÷ 2

256 色モード : 総ドット数

65536 色モード : 総ドット数×2

戻 り 値 戻り値はありません。

参 照 関 数 get, window

プログラム例

```
#include      <basic0.h>
#include      <graph.h>

main()
{
    static char    buf[14015 + 1];

    screen(1, 3, 1, 1);
    vpage(1);
    symbol(3, 1, "x68000", 2, 2, 2, 65535, 0);
    box(0, 0, 147, 49, 65535, 0xffff);

    get(1, 1, 146, 48, buf, sizeof(buf));

    put(200, 200, 345, 247, buf, sizeof(buf));
}
```

putc, putchar

レベル 2

書 式 #include <stdio.h>

```
int putc(c, stream) ;  
int c; /* 出力する文字 */  
FILE *stream; /* FILE 構造体へのポインタ */
```

```
int putchar(c) ;  
int c; /* 出力する文字 */
```

機 能 putc は stream で指定するストリームに 1 文字出力します。
putchar は putc(c, stdout) と同じになります。

戻 り 値 出力した文字を返します。
エラーが発生した場合は、EOF を返します。

参 照 関 数 fputc, fputchar,getc, getchar

プログラム例

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

main(int argc, char *argv[])
{
    FILE *fp;
    int c;

    if (argc == 1) {
        while ((c = getchar()) != EOF)
            putchar(c);
    }
    else {
        if ((fp = fopen(argv[1], "w")) == NULL) {
            perror(argv[1]);
            exit(1);
        }
        while ((c = getchar()) != EOF)
            putc(c, fp);
        (void)fclose(fp);
    }
    exit(0);
}
```

注：putc、putchar は fputc、fputchar 関数と同じですが、マクロ定義と関数の両方があります。

マクロ定義で使いたい場合は、stdio.h をインクルードする前に次の1行を追加してください。

```
#define MACRO
```

putch

レベル 2

書 式 `#include <conio.h>`

```
int putch(c) ;  
int c; /* コンソールに出力する文字 */
```

機 能 コンソールに文字 `c` を出力します。
改行文字 `'\n'` を、復帰改行に変換しません。
正しく復帰改行したい場合は、`'\r'` と `'\n'` を続けて出力してください。

戻 り 値 成功した場合は出力した文字 `c` を返します。失敗した場合は EOF を返します。

参 照 関 数 `cprintf`, `getch`, `getche`

プログラム例

```
#include      <conio.h>  
  
main()  
{  
    int      c;  
  
    c = getch();  
    putch(c);  
}
```

putenv

レベル 2

書 式 `#include <stdlib.h>`

```
int putenv(envstring) ;  
char *envstring; /* 環境文字列へのポインタ */
```

機 能 新しい環境変数の追加、および既存の環境変数の修正、削除を行います。
envstring は次のような形式の文字列へのポインタです。

```
varname = string
```

varname は追加、または修正する環境変数の名前で、string は環境変数の値です。
varname がすでに環境変数として存在する場合には、その string を指定した
string によって置き換え、そうでなければ新しい環境変数として追加します。
空の string を指定する (“varname=”を指定) すると、varname を環境から削
除します。

戻 り 値 正しく追加、修正削除できた場合には、0 を返します。
エラーが発生した場合 -1 を返します。

参 照 関 数 `getenv`

プログラム例

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

#define NAME   "path"

static char   buf[256];

main()
{
    char       *p;

    if ((p = getenv(NAME)) == NULL) {
        printf("%s is not found.\n", NAME);
        exit(1);
    }
    else
        printf("%s = %s\n", NAME, p);

    sprintf(buf, "%s=%s;%%usr%%games", NAME, p);
    if (putenv(buf)) {
        printf("can't put env.\n");
        exit(1);
    }
    p = getenv(NAME);
    printf("%s = %s\n", NAME, p);
    exit(0);
}
```


putl

レベル 2

書 式 #include <stdio.h>

```
int putl(lg, fp) ;
```

```
long lg; /* 書き出す値 */
```

```
FILE *fp; /* ファイル構造体へのポインタ */
```

機 能 fp で指定されたストリームに long サイズ分書き出します。

戻 り 値 書き出した long の値を返します。

エラーが発生した場合は EOF を返します。

ただし、ストリームに書き込んだデータが EOF と一致している可能性がありますので、エラーの判定は ferror 関数を使用して判定してください。

参 照 関 数 getw, putw, getl

puts

レベル 2

書 式 `#include <stdio.h>`

```
int puts(string) ;  
char *string; /* 出力する文字列へのポインタ */
```

機 能 指定した string を、標準出力 stdout へ出力します。
このとき文字列の終了を示すヌル文字'¥0'は改行文字'¥n'に置き換えられます。

戻 り 値 成功すると 0 を、失敗すると EOF を返します。

参 照 関 数 fputs, gets

プログラム例

```
#include      <stdio.h>  
#include      <stdlib.h>  
  
#define SIZE  4096  
  
static char   buf[SIZE];  
  
main()  
{  
  
    while (gets(buf) != NULL)  
        puts(buf);  
    exit(0);  
}
```

putw

レベル 2

書 式 `#include <stdio.h>`

```
int putw(sh, stream) ;
short sh; /* 書き出す値 */
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能 stream で指定されたストリームに short サイズ分書き出します。

戻 り 値 書き出した short の値を返します。

エラーが発生した場合は、EOF を返します。

参 照 関 数 `getw, getl, putl`

プログラム例

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

main()
{
    short s;
    FILE *fp;

    if ((s = getw(stdin)) == EOF) {
        if (ferror(stdin)) {
            perror("fail");
            clearerr(stdin);
        }
        else if (eof(stdin))
            printf("end of file\n");
        exit(1);
    }
    fprintf(stderr, "number = %d\n", s);
    putw(s, stdout);
    exit(0);
}
```

qsort

レベル 2

書 式 `#include <stdlib.h>`

```
void qsort(base, num, width, compare) ;  
void *base; /* 配列を指すポインタ */  
size_t num, width; /* 配列要素の数、配列要素のサイズ */  
int (*compare)(const void *, const void *) ; /*  
compare ルーチンを指すポインタ */
```

機 能

各要素が width バイトで、要素数が num 個の配列をクイックソートします。base はソートする配列を指すポインタです。compare は配列の 2 つの要素を比較し、その大小関係を返すユーザーが用意するルーチンを指すポインタです。qsort はソートの過程で、必要に応じて compare が指す関数を呼び出します。compare は 2 つの引数を受けとります。引数には配列の要素へのポインタが渡されますので、その 2 つを比較して、次の値のうちいずれかを返すようにします。

compare の戻り値	内 容
負	要素 1 は要素 2 より小さい
0	要素 1 と要素 2 は等しい
正	要素 1 は要素 2 より大きい

戻 り 値

戻り値はありません。

参 照 関 数 bsearch

プログラム例

```

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

#define MAX    12

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

    qsort(argv, argc, sizeof(char *), compare);
    while (argc--)
        printf("%s\n", *argv++);
}

static int compare(char **key1, char **key2)
{
    return (strcmp(*key1, *key2));
}

```

raise

レベル 2

書 式 #include <signal.h>

```
int raise(sig) ;  
int sig; /* 送りたいシグナル */
```

機 能 プログラムに、sig で指定されたシグナルを送ります。

sig は signal.h で定義されたマクロ SIGINT, SIGABRT, SIGTERM のいずれかを指定します。sig の意味を以下に示します。

sig の値	意 味
SIGINT	CTRL+C、BREAK キーによる割り込み
SIGABRT	異常終了
SIGTERM	プログラム終了の要求

signal 関数によって、各シグナルの処理が変更されていない場合かまたは、SIG_DFL が指定されていた場合は、raise によって次の処理が実行されます。

SIGINT 型シグナルを送ると、CTRL+C、BREAK キーを押した場合に引き起こされる処理が実行されます。

SIGABRT 型シグナルを送ると、abort 関数を呼び出したときに引き起こされる処理と同等な処理が実行されます。

signal 関数によって、SIG_IGN が指定されていた場合は、raise はエラーを返します。

signal 関数によって、なんらかの関数が指定されていた場合は、raise の呼び出しに引き続いてその関数が呼び出されます。

呼び出された関数には、引数として、raise に指定されたシグナル sig がそのまま受けわたされます。

呼び出された関数は、そのシグナルにみあった処理を実行後に、signal (sig, SIG_DFL) を行い、続いて raise (sig) を行い、デフォルト処理に移行してください。

原則として、関数からリターンしてはいけません。

もし、関数からリターンしなければならない場合、速やかに 0 以外の値を戻り値としてリターンしてください。

raise 関数は、その値を受け取って呼び出し元へ返します。

これにより、raise を、見かけ上、失敗させることができます。

戻り値 成功すると0を、失敗すると0以外を返します。

参照関数 signal, abort

プログラム例

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

int func(int sig)
{
    int res;

    res = 1;
    switch (sig) {
        case SIGINT:
            printf("SIGINTシグナルが発生しました\n");
            break;
        case SIGABRT:
            printf("SIGABRTシグナルが発生しました\n");
            break;
        case SIGTERM:
            printf("SIGTERMシグナルが発生しました\n");
            break;
        default:
            res = -1;
            break;
    }
    return (res);
}

main()
{
    signal(SIGINT, (void(*)())func);
    signal(SIGABRT, (void(*)())func);
    signal(SIGTERM, (void(*)())func);
    printf("raise(SIGINT)の戻り値 %d\n", raise(SIGINT));
    printf("raise(SIGABRT)の戻り値 %d\n", raise(SIGABRT));
    printf("raise(SIGTERM)の戻り値 %d\n", raise(SIGTERM));
}
```

rand

レベル 2

書 式 #include <stdlib.h>

int rand();

機 能 0~RAND_MAX の範囲の疑似乱数を発生します。

マクロ RAND_MAX は stdlib.h に定義されています。

srand 関数で乱数系列を初期化できます。

戻 り 値 0~RAND_MAX の範囲の疑似乱数を返します。

参 照 関 数 srand

プログラム例

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

#define MAX 128

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

    if (argc == 1)
        r = 0;
    else
        r = atoi(argv[1]);

    if (r == 0) {
        for (i = 0; i < MAX; i++)
            printf("%10d\n", rand());
    }
    else {
        for (i = 0; i < MAX; i++)
            printf("%10d\n", rand() % r);
    }
}
```


randomize

レベル 3

書 式 `#include <basic.h>`

```
void randomize(i) ;  
int i; /* 初期値 */
```

機 能 rnd 関数の乱数系列の初期化を行います。

戻 り 値 戻り値はありません。

rbrk

レベル 2

書 式 #include <stdlib.h>

void rbrk();

機 能 ピープ領域のブレイク値（領域の上限値）を初期状態に戻します。
 ピープ領域とは、malloc 関数などが返すメモリ領域の全体をいいます。

戻 り 値 戻り値はありません。

注：malloc 関数などによって割りつけられたメモリ領域を解放せずに、本関数を実行してはいけません。

read

レベル 2

書 式 #include <io.h>

```
size_t read(file, buffer, length) ;
int file; /* 読み込むファイルのファイルハンドル */
void *buffer; /* 読み込み領域を指すポインタ */
size_t length; /* 読み込みバイト数 */
```

機 能 fileで指定されたファイルハンドルから length バイトを読み込み、buffer が指す領域に格納します。

読み込み操作は、指定したファイルの現在位置から始まります。

ファイルポインタは読み込み後には、読み込んだ文字の次に進みます。

戻 り 値 実際に読み込んだバイト数を返します。

ファイルに length バイトより少ないバイトしか残っていなかった場合は、length よりも小さい値になります。

エンドオブファイル (EOF) を読み込んだ場合は 0 を返します。

またエラーが発生した場合は -1 を返し、errno に EBADF がセットされます。

errno の値	内 容
EBADF	指定したファイルが無効、またはファイルが読み出しのためにオープンされていない

ファイルをテキストモードでオープンした場合、戻り値は実際に読み込んだバイト数と等しくはなりません。

これはテキストモードでは、復帰改行 '\r\n' を改行 '\n' に変換し、その変換した改行のみを戻り値のバイト数に加算するためです。

この変換は、ファイルポインタには影響しません。

参 照 関 数 creat, fread, open, write

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <io.h>
#include <stat.h>

#define SIZE 16

main(int argc, char *argv[])
{
    char buf[SIZE];
    char *file;
    int fn;
    int n;
    file = (argc > 1) ? argv[1] : "file";

    if ((fn = open(file, O_RDONLY|O_BINARY)) == -1) {
        printf("can't open %s\n", file);
        exit(1);
    }
    while (!eof(fn)) {
        n = read(fn, buf, SIZE);
        printf("%d bytes readed.\n", n);
    }
    close(fn);
    exit(0);
}
```

客 内	arnno の値
指定したファイルが読めず、エラーメッセージが出力された。	EINVAL

create, read, open, write

読 取 関 数

realloc

レベル 2

書 式 `#include <stdlib.h>`

```
void *realloc(ptr, size) ;  
void *ptr; /* すでに割りつけられているメモリブロックへのポインタ */  
size_t size; /* 新しいメモリブロックのサイズ */
```

機 能

すでに割りつけられているメモリブロックの大きさを変更します。

ptr は malloc 関数や、realloc で返された、サイズを変更するメモリブロックのポインタを指定します。

size は新しいメモリブロックの大きさをバイト数で指定します。

大きさを変更してもメモリブロックの内容は変化しません。

ただし、新しいメモリブロックの位置は変化します。

戻 り 値

再割りつけしたメモリブロックを指すポインタを返します。

再割りつけられた領域は、どんな型のデータにも利用できるようになっていきます。

再割りつけ時にメモリブロックは移動するので、realloc の戻り値は必ずしも ptr と同じとはかぎりません。

メモリの不足などで指定した大きさのメモリブロックが確保できない場合には、NULL を返し、size が 0 であり ptr が NULL でないときはもとのメモリブロックが解放されます。

参 照 関 数

calloc, free, malloc

プログラム例

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

#define B_SIZE 256

main()
{
    int i;
    char *p;
    char *q;

    if ((p = malloc(B_SIZE)) == NULL) {
        printf("確保できませんでした。¥n");
        exit(1);
    }
    for (i = 0; i < B_SIZE; i++)
        p[i] = i;
    for (i = 1; i < 1024; i++) {
        printf("%4d block ", i);
        if ((q = realloc(p, (size_t)(i * B_SIZE))) == NULL) {
            printf("再確保できませんでした。¥n");
            break;
        }
        printf("確保しました。¥n");
        p = q;
    }
    for (i = 0; i < B_SIZE; i++)
        printf("p[%d] = %d¥n", i, p[i]);
    free(p);
    printf("解放しました。¥n");
    exit(0);
}
```

rename

レベル 2

書 式 #include <io. h>

```
int rename(oldname, newname) ;  
char *oldname; /* 旧ファイル名へのポインタ */  
char *newname; /* 新ファイル名へのポインタ */
```

機 能

oldname で指定したファイル名、またはディレクトリ名を newname で指定した名前に変更します。

oldname は既存のファイルまたはディレクトリのパス名でなければなりません。newname に既存のファイル、またはディレクトリ名を指定することはできません。

newname で異なったパス名を指定することにより、ファイルを他のディレクトリへ移すことができます。

ただし、異なったデバイスへ移動する（たとえば、ドライブ A からドライブ B へ）ことはできません。

また、ディレクトリは名前を変えることができるだけで、移動することはできません。

カレントディレクトリに係わる（先頭から一部、または全部に一致する）ディレクトリの名前を変えないでください。

もし変えると、その後の動作は保障しません。

戻 り 値

正常終了ならば正の数（0 も）を返します。

エラーならば -1 を返します。

また、エラーが発生した場合、errno と _doserrno にそれぞれ C のエラー番号と OS のエラー番号をセットします。

参 照 関 数

creat, fopen, open

プログラム例

```

#include      <stdio.h>
#include      <stdlib.h>
#include      <io.h>

main(int argc, char *argv[])
{
    if (argc < 3) {
        puts("Usage: rename oldName newName");
        exit(1);
    }
    else {
        if (rename(argv[1], argv[2]))
            puts("Rename error");
        else
            puts("Success");
    }
    exit(0);
}

```

戻り値

関連関数

repmem

レベル 2

書 式 `#include <string.h>`

```
void repmem(dest, source, size, n);
void *dest; /* コピー先領域のポインタ */
void *source; /* コピー元領域のポインタ */
size_t size; /* コピーするバイト数 */
int n; /* コピーする回数 */
```

機 能 source が指す領域の size バイトを、dest が指す領域に n 回コピーします。

戻 り 値 戻り値はありません。

プログラム例

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

main()
{
    static char    dst[] = "abcdefghijklmnopqrstuvmwxvxyz";
    static char    src[] = "x68000,";

    printf("%s\n", dst);
    repmem(dst, src, strlen(src), 2);
    printf("%s\n", dst);
}
```

rewind

レベル 2

書 式 #include <stdio.h>

```
void rewind(stream) ;  
FILE *stream; /* FILE 構造体へのポインタ */
```

機 能 ファイル stream のポインタをファイルの先頭へ移動します。
fseek 関数でも同様の働きが可能です。fseek 関数がエンドオブファイル (EOF) やエラーインジケータをクリアしないのに対し、rewind はそれらをクリアします。

戻 り 値 戻り値はありません。

参 照 関 数 fseek, ftell

プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
  
main(int argc, char *argv[])  
{  
    char *file;  
    FILE *fp;  
  
    file = (argc > 1) ? argv[1] : "file";  
    if ((fp = fopen(file, "rb")) == NULL) {  
        perror(file);  
        exit(1);  
    }  
    (void)fseek(fp, 0L, SEEK_END);  
    printf("tell = %ld\n", ftell(fp));  
    rewind(fp);  
    printf("tell = %ld\n", ftell(fp));  
    (void)fclose(fp);  
    exit(0);  
}
```

rgb

レベル3

書 式 #include <graph.h>

```
int rgb(r, g, b) ;
int r; /* 赤 */
int g; /* 緑 */
int b; /* 青 */
```

機 能 赤 r、緑 g、青 b の各成分に応じた色コードを得ます。

なお、色コードは 0~65534 の偶数となります。

r、g、b はそれぞれ、0~31 を指定します。

戻 り 値 色コードを返します。

プログラム例

```
#include <basic0.h>
#include <graph.h>

main()
{
    void test(int);
    int r, g, b;
    int d;

    b_init();
    screen(1, 3, 1, 1);
    vpage(1);
    for (r = 0; r < 16; r++) {
        for (g = 0; g < 16; g++) {
            for (b = 0; b < 16; b++) {
                for (d = 0; d < 2; d++)
                    test(rgb(r, g, b) + d);
            }
        }
    }
    b_exit(0);
}

static void test(int f)
{
    int e;

    circle(255, 255, 100, f, 0, 360, 256);
    paint(255, 255, f);
    for (e = 0; e < 200; e++)
        ;
}
```

rmdir

レベル 2

書 式 #include <direct.h>

```
int rmdir(pathname) ;  
char *pathname; /* 削除するディレクトリ名へのポインタ */
```

機 能

pathname で指定したディレクトリを削除します。
ただし、空でないディレクトリ、カレントディレクトリおよびルートディレクトリを削除することはできません。

戻 り 値

正常終了ならば正の数 (0 も) を返します。
エラーならば -1 を返します。
また、エラーが発生した場合、errno と _doserrno にそれぞれ C のエラー番号と OS のエラー番号をセットします。

参 照 関 数

chdir, mkdir

プログラム例

```
#include <stdio.h>  
#include <direct.h>  
  
#define DIR1 "¥¥tmp2"  
#define DIR2 "¥¥tmp2¥¥sub"  
  
main()  
{  
    if (mkdir(DIR1) == -1)  
        printf("create error `%s'.¥n", DIR1);  
    else {  
        printf("%s is created.¥n", DIR1);  
        if (mkdir(DIR2) == -1)  
            printf("create error `%s'.¥n", DIR2);  
        else {  
            printf("%s is created.¥n", DIR2);  
            (void)rmdir(DIR2);  
            printf("%s is removed.¥n", DIR2);  
        }  
        (void)rmdir(DIR1);  
        printf("%s is removed.¥n", DIR1);  
    }  
}
```

rmtemp, rmtmp

レベル 2

書 式 `#include <stdio.h>`

`int rmtemp( ) ;`

`int rmtmp( ) ;`

機 能

tempfile 関数はまた tmpfile 関数によって作成されたテンポラリファイルを削除します。

戻 り 値

削除したファイルの数を返します。

1つも削除できなかった場合は-1を返します。

参 照 関 数

tempfile, tmpfile, tmpnam, tmpnam

プログラム例

```
#include          <stdio.h>

main()
{
    FILE      *fp;

    if ((fp = tempfile("B:", "sample.tmp")) == NULL)
        fprintf(stderr, "can't open temp file\n");
    else
        rmtemp();
    exit(0);
}
```

rnd

レベル 3

書 式 #include <basic.h>

double rnd();

機 能 浮動小数点数値の擬似乱数を求めます。

戻 り 値 0 以上 1 未満の浮動小数点数の擬似乱数が返ります。

rstmem

レベル 2

書 式 #include <stdlib.h>

void rstmem();

機 能 ヒープ領域のブレイク値（領域の上限値）を初期状態に戻します。
ヒープ領域とは、malloc 関数などが返すメモリ領域の全体をいいます。

戻 り 値 戻り値はありません。

注：malloc 関数などによって割りつけられたメモリ領域を解放せずに、本関数を実行してはいけません。

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

int main()
{
    if (rstmem() == (char *)-1)
        printf("Not Successful");
    else
        printf("Successful");
}
```

sbrk

レベル 2

書 式 #include <stdlib.h>

```
char *sbrk(incr);  
size_t incr; /* ブレーク値に加算するバイト数 */
```

機 能 ヒープ領域のブレーク値（領域の上限値）に incr を加算し、ヒープ領域を拡張します。

incr に負の数を指定すると、その後の動作は保障されません。

戻 り 値 変更前のブレーク値を返します。
エラーが発生した場合は -1 を返し、errno に ENOMEM をセットします。

参 照 関 数 calloc, free, malloc, realloc

プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
  
main()  
{  
    if (sbrk(100) == (char *)-1)  
        printf("Not Success\n");  
    else  
        printf("Success\n");  
}
```


scanf

レベル 2

書 式 #include <stdio.h>

```
int scanf(format_string [, argument...] );
char *format_string; /* 書式文字列へのポインタ */
```

機 能

標準入力からデータを読み込み、format_string の書式に従って変換し、後に続く argument のフィールドに代入します。

また、引数 argument は format_string の各変換指定に対応する型変数のポインタです。

format_string は次の文字のうち、1 つ以上を含まなくてはなりません。

- 空白文字 (空白 ' ', タブ '\t', 改行 '\n', 垂直タブ '\v', 改頁 '\f', 復帰 '\r')
空白文字を読み飛ばします。

- 通常文字

一致した文字のみ、読み込みます。

標準入力から読み込まれた次の文字が一致しなかった場合には、scanf は終了します。

- パーセント (%)

次の書式で変換を指定します。

% [*] [length] type

- 【*】

入力を読み飛ばし、データを代入しません。

また、引数を必要としません。

- [length]

読み込むデータの最大長を 10 進数の整数で指定します。

最大長を超えたデータは切り捨てられます。

最大長より短いデータや指定したフォーマットでは変換できない文字を読み込んだ場合、空白文字を読み込んだ場合は、length よりも短い文字数を読み込みます。

[type]

変換書式を指定します。

type の値	入力の型	引数の型
d ld	10 進の整数	int または long を指すポインタ
hd	10 進の整数	short を指すポインタ
i li	整数 2、8、16、10 進すべて可	int または long を指すポインタ
hi	整数 2、8、16、10 進すべて可	short を指すポインタ
b lb	2 進の整数	int または long を指すポインタ
hd	2 進の整数	short を指すポインタ
o lo	8 進の整数	int または long を指すポインタ
ho	8 進の整数	short を指すポインタ
x lx	16 進の整数	int または long を指すポインタ
hx	16 進の整数	short を指すポインタ
u lu	符号なしの 10 進整数	unsigned int または unsigned long を指すポインタ
hu	符号なしの 10 進整数	unsigned short を指すポインタ
e f g	符号付きの浮動小数点値	float を指すポインタ
le lf lg	符号付きの浮動小数点値	double を指すポインタ

type の値	入力の型	引数の型
c	文字 通常は読み飛ばされる空白文字が、この“c”の指定によって読み込むことが可能 次の空白文字でない文字を読み込むには“%1S”を使う	入力文字列を格納できる大きさをもつ文字配列を指すポインタ
s	文字列	入力文字列とヌル文字‘\0’を格納できる大きさをもつ文字配列を指すポインタ
p	アドレスを表す 16 進数	ポインタを指すポインタ
n	読み込みは行なわない 現在までに入力から読み込んだ文字数を、引数が指す int 型の変数に代入する	int を指すポインタ

空白文字以外によって区切られている文字列を読み込むときは、大カッコ〔 〕の中の文字セットを s（文字列）型文字の代わりに使います。

対応する入力フィールドは大カッコにはない文字を発見するまで読み出します。

また、最初の文字がキャレット（^）の場合は、反対の操作を行います。

入力フィールドは残りの文字セットの最初の文字を発見するまで読み出します。

ヌル文字‘\0’を除いて文字列を格納するときは、“%nc”（n は 10 進数の整数）とします。

%c に対応する引数は文字配列を指すポインタで、n 個の文字を標準入力から読み出し、ヌル文字‘\0’はつけ加えません。

%p に対応する引数はポインタを指すポインタで、printf の%p 変換によって出力されたデータを読む込みのための書式です。

%n に対応する引数は、int を指すポインタで、現在までに入力から読み込んだ文字数が、引数のポインタへ代入されます。

このとき、入力は行なわれません。

scanf は次の場合、空白文字を読み出す前に読み込みを終了します。

- length で指定した文字数に達したとき
- 指定どおりに変換できない文字を読み出したとき
- 制御文字列と矛盾する文字を読み出したとき

●指定した文字セットにない文字を読み出したとき

戻り値

代入したフィールドの数を返します。

%*によって代入しなかったフィールドは含みません。

また、%nによる代入も含みません。

フィールドに1つも代入しなかった場合は0を返します。

読み込みの前に入力が尽きてしまった場合はEOFを返します。

参照関数

fscanf, printf, sscanf

プログラム例

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

```
main()
```

```
{
```

```
    char    str[256];
```

```
    char    c;
```

```
    long    l;
```

```
    float    d;
```

```
    scanf("%s\n", str);
```

```
    scanf("%c\n", &c);
```

```
    scanf("%ld\n", &l);
```

```
    scanf("%g", &d);
```

```
    printf("%s %c %ld %g\n", str, c, l, d);
```

```
}
```

screen

レベル 3

書式

```
#include <basic0.h>

void screen(size, mode, res, disp_sw) ;
int size; /* 表示画面サイズ */
int mode; /* 実画面・色モード (ページ) */
int res; /* 解像度 */
int disp_sw; /* グラフィック画面使用可・不可 */
```

機能

テキストおよびグラフィック画面の初期化を行います。
size には表示画面サイズを指定します。

- 0 : グラフィック 256×256、テキスト 32×16
- 1 : グラフィック 512×512、テキスト 64×32
- 2 : グラフィック 768×512、テキスト 96×32

mode には実画面・色モード (ページ) を指定します。

mode	実画面	色モード (ページ)
0	1024×1024	16 色 (0)
1	512×512	16 色 (0~3)
2	512×512	256 色 (0~1)
3	512×512	65536 色 (0)

size に 2 を指定した場合は、mode は 0 のみが指定可能です。
res には解像度を指定します。

- 0 : 低解像度
- 1 : 高解像度

size に 2 を指定した場合は、res は 1 のみが指定可能です。
disp_sw にはグラフィック画面の使用可・不可を指定します。

- 0 : グラフィック画面使用不可
- 1 : グラフィック画面初期化

disp_sw に 0 を指定するとグラフィックを非表示に、1 を指定すると表示状態になります。

screen 関数を実行すると、グラフィックのクリッピング領域は表示画面サイズと同一となり、テキスト画面のスクロール行は、画面全体となりホームポジションおよびカーソル位置は (0, 0) に設定されます。

また、テキスト画面は常に表示状態となり、スプライト画面は sp_disp (1) を実行するまで表示されません。

戻り値

戻り値はありません。

モード	画面実	モード
0	1024 × 1024	16 bit (10-2)
1	213 × 213	16 bit (10-2)
2	213 × 213	32 bit (10-2)
3	213 × 213	32 bit (10-2)

size に 2 を指定した場合は、mode は 0 を指定します。
size に 3 を指定した場合は、mode は 1 を指定します。

0 : 低解像度

1 : 高解像度

size に 2 を指定した場合は、size は 1 のみ指定可能です。
disp_sw に 2 を指定した場合は、画面の表示可能、不可を指定します。

0 : マスク画面表示不可

1 : マスク画面表示可能

setbuf

レベル 2

書 式 `#include <stdio.h>`

```
void setbuf (stream, buffer) ;
FILE  *stream; /* FILE 構造体へのポインタ */
char  *buffer; /* ユーザー定義バッファへのポインタ */
```

機 能 streamで指定するストリームが使用する入出力バッファを、システムが自動的に割りあてたものから任意にユーザーが定義したものに変更します。

buffer は新しい入出力バッファを指すポインタで、バッファのサイズは定数BUFSIZでなければいけません。

NULL を指定すると stream はバッファリングされません。

通常 stderr や stderr などにはバッファリングしませんが、setbuf によりバッファリングすることができます。

戻 り 値 戻り値はありません。

参 照 関 数 fflush, fopen, fclose

プログラム例

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

static char buf[BUFSIZ];

main(int argc, char *argv[])
{
    char *file;
    FILE *fp;
    int c;

    file = (argc > 1) ? argv[1] : "file";

    if ((fp = fopen(file, "r")) == NULL) {
        perror(file);
        exit(1);
    }
    fprintf(stderr, "buffering on\n");
    setbuf(fp, buf);
    while ((c = getc(fp)) != EOF)
        putchar(c);
    (void)fclose(fp);

    if ((fp = fopen(file, "r")) == NULL) {
        perror(file);
        exit(1);
    }
    fprintf(stderr, "buffering off\n");
    setbuf(fp, NULL);
    while ((c = getc(fp)) != EOF)
        putchar(c);
    (void)fclose(fp);

    exit(0);
}
```


setjmp

レベル 2

書 式 `#include <setjmp. h>`

```
int setjmp(env) ;
jmp_buf env; /* 現在のスタック環境をセーブする領域 */
```

機 能

現在のスタックを env にセーブします。

env は longjmp 関数に引きわたされます。

setjmp および longjmp 関数は非ローカル goto を実行し、エラーの操作などにおいて、通常の関数呼び出し、および関数からの復帰以外のシーケンスを実行します。

setjmp が呼び出されると、現在のスタック環境が env にセーブされ、逆に longjmp 関数が呼び出されると、セーブした環境のリストアが行われ、対応する setjmp 呼び出しの直後に制御が戻ります。

longjmp 関数は、setjmp を呼び出す前に呼び出してはいけません。

また、setjmp を呼び出した関数が return 文を実行した後や、関数から抜け出したりした後で longjmp 関数をしてはいけません。

ほかに、setjmp 呼び出し時と longjmp 関数呼び出し時でスーパーバイザ/ユーザーモードが異なってはいけません。

もし、上記の禁止事項が守られていない状態で longjmp 関数を行うと、その後のプログラムの動作は保障されません。

戻 り 値

通常の呼び出しによってスタック環境がセーブされた場合は 0 を返します。

longjmp 関数の呼び出しによってスタック環境がリストアされ、setjmp に制御が移行してきた場合は 0 以外 (longjmp 関数から受け渡された値) を返します。

参 照 関 数

longjmp

プログラム例

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

static jmp_buf label;

main()
{
    void func1(void);
    void func2(void);
    int r;

    if ((r = setjmp(label)) == 0)
        puts("setjmpを実行しました。");
    else if (r == 1)
        puts("longjmpによってfunc1()から来ました。");
    else if (r == 2)
        puts("longjmpによってfunc2()から来ました。");
    func1();
    func2();
    exit(0);
}

static void func1(void)
{
    static int n = 0;

    if (n++ < 3)
        longjmp(label, 1);
}

static void func2(void)
{
    static int n = 0;

    if (n++ < 2)
        longjmp(label, 2);
}
```

setmode

レベル 2

書 式 `#include <fcntl.h>`
`#include <io.h>`

```
int setmode(file, mode);
int file; /* 該当ファイルハンドル */
int mode; /* ファイルの変換モード */
```

機 能 file で指定されたファイルのファイル変換モードを変更します。
 変更できるモードは次の通りです。

mode の値	内 容
O_TEXT	テキストモードに変更する 入力時には復帰改行‘ <code>¥r/¥n</code> ’を改行‘ <code>¥n</code> ’に変換し、出力時には改行‘ <code>¥n</code> ’を復帰改行‘ <code>¥r/¥n</code> ’に変換する
O_BINARY	バイナリモードに変更する 上記の変換は行われない

通常 setmode は stdin、stdout、stderr、stdprn のデフォルトのモードを変更するために使用しますが、他のファイルに対しても使用することができます。

戻 り 値 正常に変更した場合は、前の変換モードを返します。
 エラーが発生した場合には -1 を返し errno に下記のどちらかをセットします。

errno の値	内 容
EBADF	指定したファイルが無効
EINVAL	mode の指定の誤り

参 照 関 数 creat, fopen, open

プログラム例

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

#define SIZE 512

main()
{
    int ifn, ofn;
    char buf[SIZE];
    int n;
    int c;

    ifn = fileno(stdin);
    ofn = fileno(stdout);
    setmode(ifn, O_BINARY);
    setmode(ofn, O_BINARY);
    while ((n = read(ifn, buf, SIZE)) > 0)
        write(ofn, buf, n);
}
```

通常 `setmode` は `stdin`, `stdout`, `stderr` のファイル・ハンドルにのみ使用されます。他のファイル・ハンドルに対しては使用できません。

正常に変更した場合は、前の変換モードに戻します。エラーが発生した場合には `errno` にエラーコードがセットされます。

errno の値	内容
EBADF	指定したファイル・ハンドルが無効
EINVAL	mode の指定が無効

create, lseek, open

参照関数

setmspos

レベル 3

書 式 `#include <mouse.h>`

```
int setmspos(x, y);

int x; /* マウスカーソルの X 座標 */
int y; /* マウスカーソルの Y 座標 */
```

機 能 指定した座標にマウスカーソルを移動します。
 x にはマウスカーソルの X 座標を指定します。
 y にはマウスカーソルの Y 座標を指定します。
 x、y に指定できる範囲は、msarea 関数によって設定された範囲内です。

戻 り 値 終了コードを返します。
 0 ならば正常終了、0 以外ならばエラーです。

プログラム例

```
#include <stdio.h>
#include <conio.h>
#include <basic0.h>
#include <mouse.h>
#include <graph.h>

main()
{
    int x, y;

    b_init();
    screen(2, 0, 1, 1);
    vpage(1);
    mouse(0);
    msarea(0, 0, 767, 511);
    box(0, 0, 767, 511, 15, 0xffff);
    mouse(1);
    for (;;) {
        cputs("mouse cursor area 'x' (0-767)¥r¥n");
        cputs("mouse cursor area 'y' (0-511)¥r¥n");
        cputs("mouse cursor position (x,y) ?");
        scanf("%d,%d", &x, &y);
        setmspos(x, y);
    }
    b_exit(0);
}
```

setvbuf, setnbf

レベル 2

書 式 #include <stdio.h>

```
int setvbuf(fp, buff, buftype, bufsiz) ;
void setnbf(fp) ;
```

```
FILE *fp; /* FILE 構造体へのポインタ */
char *buff; /* バッファへのポインタ */
int buftype; /* バッファリングの方法 */
size_t bufsiz; /* バッファサイズ */
```

機 能

setvbuf は、fp で指定するストリームのバッファを、最初に確保したものから buff に変更します。

bufsiz には、使用するバッファの大きさをバイト数で指定します。

buftype には、バッファリングの方法によって、次の定数を指定します。

buftype の値	内 容
_IOFBF	入出力ともバッファリング
_IOLBF	出力を行単位でバッファリング
_IONBF	バッファリングなし

buff に NULL を指定すると、自動的に bufsiz の大きさのメモリを確保してそれを使用します。

setnbf は、fp で指定するストリームのバッファリング処理を 1 文字単位で実行するように変更します。

これらの関数は、必ずファイルをオープンした直後に実行しなければなりません。

戻 り 値

setvbuf は正常ならば 0 を、エラーならば 0 以外を返します。

setnbf は、戻り値はありません。

sgn

レベル 3

書 式 #include <basic.h>

```
double sgn(val) ;
double val ;
```

機 能 数値 val の符号を調べた結果を double 型で得ます。

戻 り 値 符号によって、次の値を返します。

符号	戻り値
+	1
0	0
-	-1

signal

レベル 2

書 式 `#include <signal.h>`

```
void (* signal(sig, func))(int);  
int sig; /* シグナル値 */  
void (* func)(int); /* シグナルを受け取った時の処理を行う関数へのポ  
インタ */
```

機 能

割り込みシグナルを受け取った時の処理を設定します。
sig は signal.h で定義されたマクロ SIGINT, SIGABRT, SIGTERM のいずれかを指定します。

sig の意味を以下に示します。

sig の値	意 味
SIGINT	CTRL+C、BREAK キーによる割り込み
SIGABRT	異常終了
SIGTERM	プログラム終了の要求

SIGABRT, SIGTERM は、raise 関数によってシグナルを送らない限り発生しません。

また、CTRL+C、BREAK キーによる SIGINT シグナルでは、func を呼び出したのち、必ずそのシグナルのデフォルトの処理が行われます。

raise 関数によって送られた SIGINT シグナルでは、この signal で設定された func が行う処理のみが行われます。

func は signal.h で定義されたマクロ SIG_IGN, SIG_DFL、または関数へのポインタを指定します。

func 関数の引き数は、int が 1 つで、通常、戻り値はあってはいけません。

func の意味を次頁に示します。

func の値	内 容
SIG_IGN	割り込みシグナルを無視する
SIG_DFL	呼び出し中のプロセスを終了し、制御を OS に戻す 呼び出しプロセスによってオープンされたファイルは 全てクローズするが、バッファはフラッシュしない
関数への ポインタ	func によって指定した関数を実行する前に、まずシ グナルに対する処理を SIG_DFL にセットしてから関 数を実行し、その後は割り込みシグナルを受け取った 直後のポインタに制御を戻す 関数実行中に signal によって再設定しない限り割り 込みシグナルは、SIG_DEL として扱われる したがって、呼ばれた関数内においてシグナルをリセ ットすることが可能

戻 り 値

設定変更前のシグナルを受け取った時の処理の値を返します。
エラーが発生した時は SIG_ERR を返し、sig の値が無効であることを示す
EINVAL が errno にセットされます。

参 照 関 数

abort, exit, _exit, raise, spawnle, spawnlp, spawnnv, spawnve, spawnvp

プログラム例

```

#include      <stdio.h>
#include      <conio.h>
#include      <signal.h>
#include      <stdlib.h>
#include      <process.h>

main()
{
    void      brk(void);

    if (signal(SIGINT, brk) == SIG_ERR) {
        perror("couldn't set SIGINT");
        abort();
    }
    for (;;)
        puts("処理を中断してみてください。");
    exit(0);
}

static void  brk(void)
{
    (void)signal(SIGINT, SIG_IGN);
    flushall();
    cputs("処理を中断されました。");
    cputs("どこかキーを押してください。");
    (void)getch();
    cputs("¥033*");
    exit(1);
}

```

図 10-1

参考文献

sin, sinh

レベル 2

書 式 #include <math.h>

```
double sin(x) ;
double sinh(x) ;
double x ;
```

機 能 sin は x のサインを、sinh は x のハイパボリック (双曲線) サインを計算します。

戻 り 値 sin は x のサインを返します。

sinh は x のハイパボリックサインを返します。

引数が不正な値の場合は、errno に EDOM がセットされ返り値は不定になります。

オーバーフローが発生した場合は errno に ERANGE がセットされ、返り値は符号が付いた HUGE_VAL になります。

またエラー処理は、matherr 関数を使って行うことができます。

参 照 関 数 acos, asin, atan, atanh, cos, cosh, tan, tanh

プログラム例

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

main()
{
    double x;
    int i;

    for (i = -32; i <= 32; i++) {
        x = (double)i / 10.0;
        printf("sin(%+4.1f) = %g\n", x, sin(x));
    }
    for (i = -32; i <= 32; i++) {
        x = (double)i / 10.0;
        printf("sinh(%+4.1f) = %g\n", x, sinh(x));
    }
}
```

sizmem

レベル 2

書 式 #include <stdlib.h>

long sizmem();

機 能

使用可能なメモリブロックに関する情報を通知します。

使用可能なメモリブロックの残りのサイズを、ワード数で通知します。

戻 り 値

使用可能なメモリブロックの残りのサイズを返します。

プログラム例

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

main()
{
    long    size;

    size = sizmem();
    printf("使用可能メモリブロックは %ldワード\n",size);
}
```

spawnl, spawnle, spawnlp, spawnv, spawnve, spawnvp

レベル 2

書 式 #include <process.h>

```
int spawnl(mflag, path, arg0, arg1, arg2, ..., argn, NULL) ;
int spawnle(mflag, path, arg0, arg1, arg2, ..., argn, NULL, envp) ;
int spawnlp(mflag, path, arg0, arg1, arg2, ..., argn, NULL) ;
int spawnv(mflag, path, argv) ;
int spawnve(mflag, path, argv, envp) ;
int spawnvp(mflag, path, argv) ;
int mflag ;
char *path ;
char *arg0, *arg1, *arg2, ..., *argn ;
char *argv [ ], *envp [ ] ;
```

機 能

子プロセスを新たに作成して実行します。

このとき、子プロセスを実行するための十分なメモリが必要です。

mflag の値によって、実行中の動作を決定します。

mflag の値と内容は次の通りです。

mflag の値	内 容
P_WAIT	子プロセスが終了するまで、親プロセスの動作を保留する

path には子プロセスとして実行するファイル名を指定します。

拡張子を省略すると、'X'をつけます。

spawnlp と spawnvp は、環境変数 path で設定したディレクトリからファイルを探します。

子プロセスへの引数は、文字列を指すポインタかポインタの配列としてわたします。

パス名と引数をあわせて 255 バイトまでで指定します。

文字列末のヌル文字もこのバイト数に含まれますので注意してください。

子プロセスへの引数の数があらかじめわかっているときは、spawnl, spawnle, spawnlp を使います。

引数の数がわからない、つまり変数のときは、spawnv, spawnve, spawnvp を使います。

spawnl, spawnlp, spawnv, spawnvp は、親プロセスの環境を引き継ぎます。

spawnle, spawnve では、envp を使って子プロセスの環境設定を変えることがで

きます。

envp は文字列を指すポインタの配列で、各文字列は、

VALNAME = value

とします。

ここで VALNAME は環境変数名で、value はその値です。

envp が NULL のときは、親プロセスの環境を引き継ぎます。

戻り値

正常終了した場合は正の数 (0 も) を返します。

これは起動したプロセスの終了コードです。

エラーが発生した場合は -1 を返します。

その場合 errno に下記の値がセットされます。

errno の値	内 容
E2BIG	引数リストが 256 バイトを超えている
EMFILE	オープンファイルの数が多すぎる
ENOENT	ファイルまたはパス名が見つからない
ENOEXEC	指定のファイルが実行できない
ENOMEM	子プロセスを実行するためのメモリが不足している

参照関数

execl, execlp, execl, execlp, execv, execve, execvp, execvpe, abort, exit

sp_clr

レベル 3

書 式 #include <sprite. h>

```
void sp_clr(cd1, cd2) ;
int cd1, cd2; /* スプライトプレーン番号 */
```

機 能

スプライトをクリアします。

cd1、cd2 でクリアするスプライトプレーンの範囲を指定します (いずれも 0 ~255)。

cd1 に 'NASI' を指定すると、cd1 は 0 になります。

cd2 に 'NASI' を指定すると、cd1 のみの指定になります。

cd1、cd2 ともに、'NASI' を指定すると cd1 は 0、cd2 は 255 になります。

戻 り 値

戻り値はありません。

sp_color

レベル 3

書 式 #include <sprite. h>

```
int sp_color(p, c, pb) ;  
char p; /* パレットコード */  
int c; /* カラーコード */  
int pb; /* パレットブロック */
```

機 能 スプライトのパレットの色の設定および読み出しを行います。
p にはパレットコードを指定します (0~15)。
c にはカラーコードを指定します (0~65535)。
c に 'NASI' を指定すると、カラーコードの読み出しのみを行います。
pb にはパレットブロックを指定します (1~15)。
pb に 'NASI' を指定すると、1 を指定したことになります。

戻 り 値 設定前のカラーコードを返します。

sp_def

レベル 3

書 式 #include <sprite. h>

```
void sp_def(cd, ca, psize) ;  
char cd; /* パターンコード */  
char *ca; /* パターンデータを格納してある領域へのポインタ */  
int psize; /* パターンデータのサイズ */
```

機 能 スプライトにパターンを定義します。

cd にはパターンコードを指定します (0~255)。

ca には、パターンデータを格納してある領域へのポインタを指定します。

この領域は、パターンサイズが8×8のときは64バイト、パターンサイズが16×16のときは256バイト必要です。

格納しておくデータの1バイトが、スプライトパターンの1ドットに対応します。

0~15のデータを格納しておくと、それがスプライトのドットの色になります。

psize にはパターンデータのサイズを指定します (0、1)。

0を指定するとパターンサイズが8×8に、1を指定するとパターンサイズが16×16になります。

‘NASI’を指定すると、1を指定したことになります。

戻 り 値 戻り値はありません。

sp_disp

レベル 3

書 式 #include <sprite.h>

```
void sp_disp(ch) ;  
char ch; /* 表示モード */
```

機 能 スプライト面（スプライト、バックグラウンド）の表示制御を行います。
ch には次の内容を指定します。

0 : スプライト、バックグラウンドを表示しない

1 : スプライト、バックグラウンドを表示する

戻 り 値 戻り値はありません。

sp_init

レベル 3

書 式 #include <sprite. h>

int sp_init();

機 能 スプライト画面を初期化します。
初期化するものは次の通りです。

スクロールレジスタ

スプライトパターン

スプライト用パレット

戻 り 値 正常に初期化できたときは、0 を返します。
エラーのとき（画面サイズが 768×512 のときなど）は -1 を返します。

sp_move

レベル 3

書 式 #include <sprite.h>

```
void sp_move(s, x, y, cd) ;  
char s; /* プレーン番号 */  
int x; /* X座標 */  
int y; /* Y座標 */  
int cd; /* パターンコード */
```

機 能 スプライトパターンをプレーンに登録し、表示位置を決めます。
sにはプレーン番号を指定します (0~127)。
xにはX座標を指定します (-16~1007)。
yにはY座標を指定します (-16~1007)。
cdにはパターンコードを指定します (0~255)。
パターンコードを指定すると、そのスプライトは自動的に表示されます。
x、y、cdに'NASI'を指定すると、以前に設定されていた値を使用します。

戻 り 値 戻り値はありません。

プログラム例

```

#include      <math.h>
#include      <basic0.h>
#include      <sprite.h>

static char   pat2[256];
static char   pat1[256] = {
    0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
    0, 0, 2, 2, 2, 3, 3, 3, 3, 3, 3, 2, 2, 2, 0, 0, 0,
    0, 2, 4, 4, 4, 5, 5, 5, 5, 5, 5, 4, 4, 4, 2, 0, 0,
    1, 2, 4, 6, 6, 6, 6, 7, 7, 7, 7, 6, 6, 6, 4, 2, 1,
    1, 2, 4, 6, 8, 8, 9, 9, 9, 9, 8, 8, 6, 4, 2, 1,
    1, 3, 5, 6, 8, 10, 11, 11, 11, 11, 10, 8, 6, 5, 2, 1,
    1, 3, 5, 7, 9, 11, 12, 13, 13, 12, 11, 9, 7, 5, 3, 1,
    1, 3, 5, 7, 9, 11, 13, 15, 15, 13, 11, 9, 7, 5, 3, 1,
    1, 3, 5, 7, 9, 11, 13, 15, 15, 13, 11, 9, 7, 5, 3, 1,
    1, 3, 5, 7, 9, 11, 12, 13, 13, 12, 11, 9, 7, 5, 3, 1,
    1, 3, 5, 6, 8, 10, 11, 11, 11, 11, 10, 8, 6, 5, 2, 1,
    1, 2, 4, 6, 8, 8, 9, 9, 9, 9, 8, 8, 6, 4, 2, 1,
    1, 2, 4, 6, 6, 6, 7, 7, 7, 7, 6, 6, 6, 4, 2, 1,
    0, 2, 4, 4, 4, 5, 5, 5, 5, 5, 5, 4, 4, 4, 2, 0, 0,
    0, 0, 2, 2, 2, 3, 3, 3, 3, 3, 3, 2, 2, 2, 0, 0,
    0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
};

main()
{
    int      x;
    int      y;
    int      i;

    b_init();
    screen(1, 3, 1, 1);
    sp_clr(0, 255);
    sp_off(0, 127);
    sp_disp(1);
    sp_def(0, pat1, 1);
    sp_def(1, pat2, 1);
    for (i = 0; i < 256; i++) {
        x = i * 2 - 16;
        y = sin(2.0 * 3.1416 * x / 256) * 200.0;
        sp_move(0, x, 255 - y, 0);
        sp_move(1, x, 255 + y, 1);
    }
    sp_disp(0);
    b_exit(0);
}

```

sp_off

レベル 3

書 式 #include <sprite. h>

```
void sp_off(s1, s2) ;
int s1, s2; /* スプライトプレーン番号 */
```

機 能 スプライトを非表示状態にします。

s1 から s2 までのプレーンのスプライトを非表示状態にします (いずれも 0 ~127)。

s1 に 'NASI' を指定すると、s1 は 0 になります。

s2 に 'NASI' を指定すると、s1 のみの指定になります。

s1, s2 ともに 'NASI' を指定すると、s1 は 0、s2 は 127 になります。

戻 り 値 戻り値はありません。

sp_on

レベル 3

書 式 #include <sprite.h>

```
void sp_on(s1, s2) ;  
int s1, s2; /* スプライトプレーン番号 */
```

機 能 スプライトを表示状態にします。
s1からs2までのプレーンのスプライトを表示状態にします(いずれも0~127)。
s1に'NASI'を指定すると、s1は0になります。
s2に'NASI'を指定すると、s1のみの指定になります。
s1、s2ともに'NASI'を指定すると、s1は0、s2は127になります。

戻 り 値 戻り値はありません。

sp_pat

レベル 3

書 式 #include <sprite.h>

```
void sp_pat(cd, ca, psize) ;  
char cd; /* パターンコード */  
char *ca; /* パターンデータを格納する領域へのポインタ */  
int psize; /* パターンデータのサイズ */
```

機 能

スプライトのパターンデータを指定した領域へ読み込みます。

cd にはパターンコードを指定します (0~255)。

ca にはパターンデータを読み込む領域へのポインタを指定します。

この領域は、パターンサイズが8×8のときは64バイト、パターンサイズが16×16のときは256バイト必要です。

スプライトパターンの1ドットが1バイトのデータになって格納されます。

スプライトのドットの色が0~15のデータになります。

psize にはパターンデータのサイズを指定します (0, 1)。

0を指定するとパターンサイズが8×8に、1を指定するとパターンサイズが16×16になります。

'NASI' を指定すると、1を指定したことになります。

戻 り 値

戻り値はありません。

プログラム例

```

#include      <stdio.h>
#include      <basic0.h>
#include      <graph.h>
#include      <sprite.h>

#define SIZE  (16 * 16)

static char  pat2[SIZE];
static char  pat1[SIZE];

main()
{
    int      x;
    int      y;
    int      i;

    b_init();
    for (i = 0; i < SIZE; i++) {
        pat1[i] = i / 32;
        pat2[i] = i / 32 + 8;
    }
    screen(1, 3, 1);
    sp_clr(0, 255);
    sp_off(0, 127);
    sp_disp(1);
    sp_def(0, pat1, 1);
    sp_def(1, pat2, 1);
    for (i = 0; i < 16; i++)
        sp_color((char)i, rgb(i * 2, i * 2, i * 2), 1); /* ! */
    sp_move(0, 100, 100, 0);
    sp_move(1, 100, 200, 1);
    sp_pat(0, pat2, 1);
    for (y = 0; y < 16; y++) {
        for (x = 0; x < 16; x++)
            printf("%d ", pat2[x + y * 16]);
        putchar('\n');
    }
    sp_disp(0);
    b_exit(0);
}

```

sprintf

レベル 2

書 式 #include <stdio.h>

```
int sprintf(buffer, format_string [, argument...] ) ;  
char *buffer; /* 出力データ格納領域へのポインタ */  
char *format_string; /* 書式文字列へのポインタ */
```

機 能 format_string で指定した書式に従って出力を変換し、buffer が指す領域に格納します。
format_string に続く引数は書式に従って変換され、出力されます。
format_string の書式と機能は printf 関数と同じです。
詳細については、printf 関数を参照してください。

戻 り 値 出力した文字数を返します。

参 照 関 数 cprintf, fprintf, printf, scanf

プログラム例

```
#include <stdio.h>  
  
static char buf[256];  
  
main()  
{  
    sprintf(buf, "hello world");  
    puts(buf);  
    sprintf(buf, "integer[%d]", 200);  
    puts(buf);  
    sprintf(buf, "integer[%x]", 200);  
    puts(buf);  
    sprintf(buf, "float[%f]", 3.1415926);  
    puts(buf);  
    sprintf(buf, "float[%g]", 3.1415926);  
    puts(buf);  
    sprintf(buf, "character[%c]", 'A');  
    puts(buf);  
    sprintf(buf, "string[%s]", "string");  
    puts(buf);  
}
```

sp_set

レベル 3

書 式 `#include <sprite.h>`

```
int sp_set(c, c1, c2, pcode, prty) ;
char c; /* スプライトプレーン番号 */
int x; /* X座標 */
int y; /* Y座標 */
int pcode; /* パターンコード */
int prty; /* プライオリティ */
```

機 能

スプライトのスクロールレジスタの設定を行います。

c にはプレーン番号を指定します (0~127)。

x には X 座標を指定します (0~1023)。

y には Y 座標を指定します (0~1023)。

pcode には、反転の指定、パレットブロック番号指定、パターンコードの指定をします。

pcode はビットごとに次の内容と値を設定できます。

ビット 16~31 : 常に 0

ビット 15 : 垂直反転 (0…標準、1…反転)

ビット 14 : 水平反転 (0…標準、1…反転)

ビット 12~13 : 常に 0

ビット 8~11 : パレットブロック番号 (0~15)

ビット 0~7 : パターンコード (0~255)

prty には表示の優先度を指定します (0~3)。

0 : スプライト表示なし

1 : バックグラウンド 0、バックグラウンド 1、スプライトの順

2 : バックグラウンド 0、スプライト、バックグラウンド 1 の順

3 : スプライト、バックグラウンド 0、バックグラウンド 1 の順

c1, c2, pcode, prty に 'NASI' を指定すると、以前に設定されていた値を使用します。

戻 り 値

正常終了したときは 0 を返します。

エラーが発生した (画面サイズが 768×512 のときなど) ときは -1 を返します。

参 照 関 数

sp_stat

sp_stat

レベル 3

書 式 #include <sprite. h>

```
int sp_stat(sp, sel_mode) ;  
char sp; /* スプライトプレーン番号 */  
char sel_mode; /* モード */
```

機 能 指定したスプライトのスクロールレジスタの状態を調べます。
sp にはスプライトプレーン番号を指定します。
sel_mode には調べる内容を指定します。

- 0 : X 座標
- 1 : Y 座標
- 2 : パターンコード
- 3 : プライオリティ

X 座標、Y 座標、パターンコード、プライオリティについての詳細は sp_set を参照してください。

戻 り 値 正常終了のときは sel_mode に従い、調べた値（正の数）を返します。
エラーが発生したときは負の数を返します。

参 照 関 数 sp_set

sqrt

レベル 2

書 式 `#include <math.h>`

```
double sqrt(x) ;
double x; /* 正の浮動小数点値 */
```

機 能 x の平方根を計算します。

戻 り 値 x の平方根を返します。
引数が不正な値の場合は、errno には EDOM がセットされ戻り値は不定になります。
またエラー処理は、matherr 関数ルーチンを使って行うことができます。

参 照 関 数 exp, log, matherr, pow

プログラム例

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

main()
{
    double x;
    int i;

    for (i = 0; i <= 32; i++) {
        x = (double)i / 10.0;
        printf("sqrt(%+4.1f) = %g\n", x, sqrt(x));
    }
}
```

srand

レベル 2

書 式 `#include <stdlib.h>`

```
void srand(seed) ;  
unsigned int seed;
```

機 能 擬似乱数の発生系列を初期化します。
再初期化する場合は、seed を 1 にします。
seed を 1 以外にすると、その値に対応した初期化が行われます。

戻 り 値 戻り値はありません。

参 照 関 数 rand

プログラム例

```
#include      <stdio.h>  
#include      <stdlib.h>  
  
#define MAX    128  
#define RANGE  100  
  
main(int argc, char *argv[])  
{  
    int    i;  
    int    s;  
  
    s = (argc == 1) ? 1 : atoi(argv[1]);  
  
    srand(s);  
  
    for (i = 0; i < MAX; i++)  
        printf("%10d\n", rand() % RANGE);  
}
```

sscanf

レベル 2

書 式 `#include <stdio.h>`

```
int sscanf(buffer, format_string [, argument...] );
char *buffer; /* 読み込まれる文字列へのポインタ */
char *format_string; /* 書式文字列へのポインタ */
```

機 能

buffer が指す文字列を読み込み、format_string の書式に従って変換し、後に続く argument のフィールドに代入します。
format_string の内容は scanf と同じです。
また、引数 argument は format_string の各変換指定に対応する型変数のポインタです。

戻 り 値

代入したフィールドの数を返します。
代入しなかったフィールドは含みません。
代入したフィールドがない場合は 0 を返します。
読み込みの前に入力 that 尽きてしまった場合は EOF を返します。

参 照 関 数 fscanf, scanf, sprintf

プログラム例

```
#include <stdio.h>

static char buf[256];

main()
{
    char str[256];
    char c;
    long l;
    float d;

    gets(buf);
    sscanf(buf, "%s\n", str);
    gets(buf);
    sscanf(buf, "%c\n", &c);
    gets(buf);
    sscanf(buf, "%ld\n", &l);
    gets(buf);
    sscanf(buf, "%g", &d);

    printf("%s %c %ld %g\n", str, c, l, d);
}
```

stat

レベル 2

書 式 #include <stat.h>

```
int stat(pathname, buffer) ;
char *pathname; /* 情報を得たいファイルのパス名 */
struct stat *buffer; /* stat 構造体を指すポインタ */
```

機 能 pathname で指定したファイル、またはディレクトリに関する情報を buffer が指す構造体に格納します。

stat 構造体は stat.h で以下のように定義されています。

```
struct stat {
    dev_t  st_dev; /* ファイルが存在するドライブの番号 */
    ino_t  st_ino; /* 常に 0 */
    short  st_mode; /* ファイルタイプとファイル属性
                     ディレクトリならば S_IFDIR のビットが
                     セットされ、ファイルならば S_IFREG の
                     ビットがセットされる
                     アクセスモードによってユーザー（リー
                     ド/ライト）のビットがセットされる
                     実行可能ファイル（.X .R .Z .BAT）なら
                     ば S_IEXEC のビットがセットされる
                     */
    short  st_nlink; /* 常に 1 */
    short  st_uid; /* 常に 0 */
    short  st_gid; /* 常に 0 */
    dev_t  st_rdev; /* ファイルが存在するドライブの番号 */
    off_t  st_size; /* ファイルの大きさ（バイト数） */
    time_t st_atime; /* 最終変更日時。time 関数の返回值と同じ
                     表現 */
    time_t st_mtime; /* 最終変更日時。time 関数の返回值と同じ
                     表現 */
    time_t st_ctime; /* 最終変更日時。time 関数の返回值と同じ
                     表現 */
};
```

戻 り 値 ステータス情報を正しく得た場合には 0 を返します。

エラーが発生した場合は -1 を返し、ENOENT を errno にセットします。

参照関数 access, fstat

プログラム例

```
#include      <stdio.h>
#include      <stdlib.h>
#include      <stat.h>

main(int argc, char *argv[])
{
    struct stat    inf;
    char    *file;

    file = (argc > 1) ? argv[1] : "file";
    if (stat(file, &inf) == -1) {
        perror(file);
        exit(1);
    }

    printf("name = %s, size = %d\n", file, inf.st_size);
    exit(0);
}
```

stcgfe, stcgfn

レベル 2

書 式 #include <string.h>

```
int stcgfe(ext, name) ;  
char *ext; /* 拡張子を格納する領域へのポインタ */  
char *name; /* ファイル名へのポインタ */  
  
int stcgfn (node, name) ;  
char *node; /* 主ファイル名を格納する領域へのポインタ */  
char *name; /* ファイル名へのポインタ */
```

機 能 ファイル名の解析を行います。

stcgfe は、name が指すファイル名から拡張子を取り出し、ext に格納してその文字数を返します。

stcgfn は、name が指すファイル名から主ファイル名（'.'と拡張子を除いたもの）を取り出し、node に格納してその文字数を返します。

戻 り 値 拡張子、または主ファイル名の文字数を返します。

stick

レベル 3

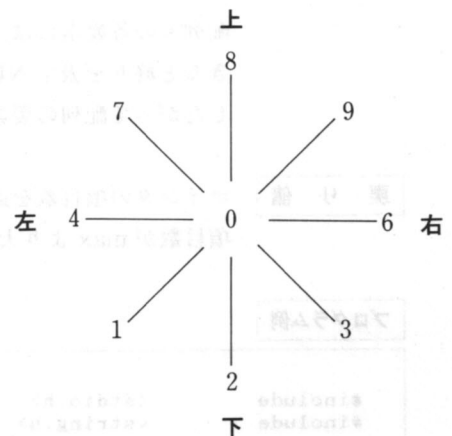
書 式 `#include <stick. h>`

```
int stick(stickno) ;
int stickno; /* ジョイスティック番号 */
```

機 能 ジョイスティックの状態を調べます。
stickno に 1 を指定するとジョイスティック 1 を、2 を指定するとジョイスティック 2 を調べます。

戻 り 値 状態によって、次の値を返します。

スティック位置	戻り値
中央	0
左後	1
後	2
右後	3
左	4
右	6
左前	7
前	8
右前	9



[ジョイスティックの状態と戻り値]

strbpl

レベル 2

書 式 #include <string.h>

```
int strbpl(s, max, t)
char *s [ ]; /* ポインタ配列 */
int max; /* 配列の要素数 */
char *t; /* 文字列を指すポインタ */
```

機 能

tが指す文字列の並びをもとに、ポインタ配列をsに作成します。

tは、それぞれヌル文字'¥0'で終了する文字列の並びであり、tの最後にはもう1つヌル文字がなければなりません。

maxには、配列sの要素数を指定します。

tに含まれる文字列の数がmaxより大きい場合、strbplは-1を返します。

配列sの各要素には、tに含まれる文字列へのポインタが代入され、文字列が尽きると終りを表すNULLを要素に代入します。

したがって配列の要素数は文字列の数より1つ多くなければなりません。

戻 り 値

ポインタの項目数を返します。

項目数がmaxより大きいときには-1を返します。

プログラム例

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

#define MAX 16

char t[] = "Sharp¥0Computer¥0x68000¥0C compiler¥0PRO 68K¥0";

main()
{
    char *s[MAX];
    int i;
    int n;

    n = strbpl(s, MAX, t);

    for (i = 0; i < n; i++)
        printf("%s¥n", s[i]);
}
```

strcat, strchr, strcmp, strcmpi, stricmp, strcpy, strcspn, strdup

レベル 2

書 式

#include <string.h>

char *strcat(string1, string2);

char *string1; /* 連結される文字列を指すポインタ */

char *string2; /* 連結する文字列を指すポインタ */

char *strchr(string, c);

char *string; /* 検索が行われる文字列を指すポインタ */

int c; /* 検索する文字 */

int strcmp(string1, string2);

char *string1, *string2; /* 比較が行われる2つの文字列を指すポインタ */

int strcmpi(string1, string2);

char *string1, *string2; /* 比較が行われる2つの文字列を指すポインタ */

int stricmp(string1, string2);

char *string1, *string2; /* 比較が行われる2つの文字列を指すポインタ */

char *strcpy(string1, string2);

char *string1; /* 複写先文字列を指すポインタ */

char *string2; /* 複写元文字列を指すポインタ */

size_t strcspn(string1, string2);

char *string1; /* 検索が行われる文字列を指すポインタ */

char *string2; /* 文字セットを指すポインタ */

char *strdup(string);

char *string; /* 複写を行う文字列を指すポインタ */

参 照

機

能

末尾がヌル文字'¥0'の文字列に対して次の操作を行います。

【strcat】

string1 のうしろに string2 を連結して 1 つの文字列にして、末尾にヌル文字'¥0'を付加します。

【strchr】

string 内で文字 c を検索します。

なお、ヌル文字'¥0'も検索することができます。

【strcmp】

string1 と string2 を文字コードの順序で比較します。

大文字と小文字は区別されます。

【strcmpi】

string1 と string2 を文字コードの順序で比較します。

大文字と小文字は区別されません。

【stricmp】

string1 と string2 を文字コードの順序で比較します。

大文字と小文字は区別されません。

【strcpy】

文字列 string2 (末尾のヌル文字を含む) を文字列を格納する領域 string1 にコピーします。

【strcspn】

string1 先頭から、string2 に含まれない文字が何文字続いているかを調べます。

【strdup】

領域を自動的に割りつけて、その領域に string をコピーします。

戻り値

以下の戻り値を返します。

【strcat】

連結された文字列を指すポインタを返します。

【 strchr 】

文字 c の位置を指すポインタを返します。

文字が見つからなかった場合は NULL を返します。

【 strcmp 】

戻り値とその内容は次の通りです。

戻り値	内 容
負	string1 は string2 より小さい
0	string1 と string2 は等しい
正	string1 は string2 より大きい

【 strcmpi 】

strcmp と同じです。

【 stricmp 】

strcmp と同じです。

【 strcpy 】

string1 を指すポインタを返します。

【 strcspn 】

何文字続いていたかをバイト数で返します。

最初から string2 に含まれる文字があれば 0 を返します。

すべてが string2 に含まれていない文字ならば、string1 の全長（ヌル文字は含まない）を返します。

【 strdup 】

割りつけられた領域を指すポインタを返します。

割りつけができなかった場合には NULL を返します。

プログラム例

```

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

static char  p[256];

main()
{
    printf("p\n");
    strcpy(p, "Sharp");
    printf("p\n");
    strcat(p, "x68000");
    printf("strchr(p, '6')\n");
    strchr(p, '6');
    printf("strcmp(p, ¥\"sharp¥\")\n");
    strcmp(p, "sharp");
    printf("strcmpi(p, ¥\"sharp¥\")\n");
    strcmpi(p, "sharp");
    printf("strcspn(p, ¥\"0123456789¥\")\n");
    strcspn(p, "0123456789");
}

```


strerror

レベル 2

書 式 #include <string.h>

```
char *strerror(errnum);
int errnum; /* エラー番号 */
```

機 能

エラー番号 errnum に対応するエラーメッセージへのポインタを返します。

errnum は、error.h において定義されている番号のみが指定できます。

strerror はマクロで定義されていますので、返される文字列を書きかえてはいけません。

返された文字列を加工したい場合は、いったん他のバッファへコピーしてから、コピーしたものを加工してください。

戻 り 値

エラーメッセージへのポインタを返します。

定義されていないエラー番号を指定すると、戻り値は不定です。

プログラム例

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

main()
{
    void sub(FILE *, FILE *);

    sub(stdin, stdout);
}

static void sub(FILE *fp, FILE *ofp)
{
    int c;

    while ((c = getc(fp)) != EOF)
        putc(c, ofp);
    if (ferror(ofp)) {
        fprintf(stderr, "Error: %s\n", strerror(errno));
        clearerr(ofp);
    }
}
```

strftime

レベル 2

書 式 #include <time.h>

```
size_t strftime(buf, max, format, time_ptr) ;  
char *buf; /* 文字列領域へのポインタ */  
size_t max; /* 作成する文字列領域のバイト数 */  
char *format; /* 書式制御文字列へのポインタ */  
struct tm *time_ptr; /* 変換される時間情報へのポインタ */
```

機 能 time_ptr が指す構造体に分解された時間情報を、format が指す書式にしたがって文字列に変換し、buf が指す領域に作成します。
max は作成する文字列領域のバイト数を指定します。
max は終端のヌル文字'¥0'を含めたバイト数を指定するので、返される文字列の最大長は max-1 になります。
format には、以下に示す書式制御文字があります。

書式	意 味
%A	曜日の名前
%a	省略された曜日の名前
%B	月の名前
%b	省略された月の名前
%c	ctime 関数の返す形式と同じ形式の曜日、日付、時間。
%d	日を表す 01~31 の数字
%H	24 時間制での時間を表す 00~23 の数字
%I	12 時間制での時を表す 00~12 の数字
%j	1 月 1 日からの通算日数を表す 001~366 の数字
%M	分を表す 00~59 の数字
%m	月を表す 01~12 の数字
%p	午前、午後を表す AM または PM の 2 文字
%S	秒数を表す 00~59 の数字

書式	意 味
%U	日曜を週の初めとする 1 月 1 日からの通算週を表す 00～53 の数字
%W	月曜を週の初めとする 1 月 1 日からの通算週を表す 00～53 の数字
%w	日曜を 0 とする曜日を表わす 0～6 の数字
%X	時：分：秒で表す時刻
%x	月/日/年で表す日付
%Y	年を表す世紀部分を付け加えた 1980～2079 の数字
%y	年を表す世紀部分を取り除いた 00～99 の数字
%Z	地域（時間帯）を表す JST の文字列
%%	%1 文字

戻 り 値

変換した文字列の長さを返します。

0 が返された場合は、変換結果が max を超えてしまう場合で、文字列領域の内容は不定になります。

参 照 関 数

asctime, ctime, gmtime, localtime

プログラム例

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

#define ARB 128
#define FORM1 "%a %b %d %H:%M:%S %Z %Y"
#define FORM2 "%B %m %y (%H:%M:%S)"
#define FORM3 "%Y/%m/%d %H:%M:%S"
#define FORM4 "%y年%m月%d日 %pI時%M分%S秒"
```

```
main()
{
    char buf[ARB];
    time_t tim;
    struct tm *tp;

    time(&tim);
    tp = localtime(&tim);

    strftime(buf, ARB, FORM1, tp);
    printf("%s\n", buf);
    strftime(buf, ARB, FORM2, tp);
    printf("%s\n", buf);
    strftime(buf, ARB, FORM3, tp);
    printf("%s\n", buf);
    strftime(buf, ARB, FORM4, tp);
    printf("%s\n", buf);
}
```

strig

レベル 2

書 式 #include <stick.h>

```
int strig(no) ;
int no; /* ジョイスティック番号 */
```

機 能 ジョイスティックのトリガボタンの状態を調べます。

no には、次の値を指定します。

- 1 : ジョイスティック 1
- 2 : ジョイスティック 2

戻 り 値 エラーのときは-1を、そうでなければ、0 から 3 の値を返します。
それぞれの値の意味は次の通りです。

- 0 : どちらのボタンも押されていない
- 1 : 1 のボタンだけ押されている
- 2 : 2 のボタンだけ押されている
- 3 : 両方とも押されている

プログラム例

```
#include <stdio.h>
#include <basic0.h>
#include <stick.h>

main()
{
    int r;

    b_init();
    cls();

    for (;;) {
        locate(10, 10);
        puts("push trigger!");
        r = strig(1);
        locate(10, 11);
        if (r == 0)
            ;
        else if (r == 1)
            puts("'A' trigger on!");
        else if (r == 2)
            puts("'B' trigger on!");
        else if (r == 3) {
            puts("'A' & 'B' triggers on!");
            break;
        }
    }
    b_exit(0);
}
```

strins

レベル 2

書 式 #include <string.h>

```
void strins(source, dest) ;  
char *source; /* 挿入する文字列へのポインタ */  
char *dest; /* 挿入先文字列へのポインタ */
```

機 能 dest が指す文字列の前に、source が指す文字列を挿入し、1つの文字列とします。

dest が指す文字列の領域は十分な大きさがなければなりません。

戻 り 値 戻り値はありません。

プログラム例

```
#include <stdio.h>  
#include <string.h>  
  
static char dst[256] = "x68000";  
  
main()  
{  
    puts(dst);  
    strins("Computer ", dst);  
    puts(dst);  
    strins("Sharp ", dst);  
    puts(dst);  
}
```

strlen

レベル 2

書 式 #include <string. h>

```
size_t strlen(str) ;  
char *str; /* 文字列 */
```

機 能 文字列 str の長さをバイト数で求めます。
このとき、文字列の最後のヌル文字'¥0'は長さを含めません。

戻 り 値 str の長さを返します。

strlwr

レベル 2

書 式 #include <string. h>

```
char *strlwr(str) ;
char *str; /* 文字列 */
```

機 能 文字列 str のすべての英大文字を英小文字に置き換えます。
他の文字は変換しません。

戻 り 値 str を指すポインタを返します。

参 照 関 数 strupr

strmfe

レベル 2

書 式 `#include <string.h>`

```
void strmfe(new, old, ext);
char *new; /* 新ファイル名格納領域へのポインタ */
char *old; /* 旧ファイル名へのポインタ */
char *ext; /* 置き換える拡張子へのポインタ */
```

機 能 与えられた要素から、次の形式のファイル名を組み立てます。

drive:¥path¥node.ext

strmfe は、old が指すファイル名の拡張子を、ext が指す拡張子で置き換え、new が指す領域に格納します。

戻 り 値 戻り値はありません。

プログラム例

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

static char new[256];
static char old[] = "sample.c";
static char ext[] = "cc";

main()
{
    printf("Old = %s\n", old);
    strmfe(new, old, ext);
    printf("New = %s\n", new);
}
```

strmf, strmf

レベル 2

書 式 #include <string.h>

```
void strmf(fname, drv, path, node, ext) ;  
char *fname, *drv, *path, *node, *ext ;  
  
void strmf(fname, path, node) ;  
char *fname, *path, *node ;
```

機 能 strmf は、drv、path、node、ext の各引数からファイル名を組み立てて、fname に格納します。
引数との対応は次の通りです。

drv : ドライブ名
path : パス名
node : 主ファイル名
ext : 拡張子

strmf では、それぞれの引数が NULL かどうかを調べます。

NULL でなければ、drv の後には ":" を、path の後には "¥" を加えます。

また、node が NULL でなければ、node の後に "." を加えます。

strmf は、path と node からファイル名を組み立てて、fname に格納します。

引数との対応は次の通りです。

path : パス名 (ドライブ名を含む)
node : 主ファイル名

もし、path が NULL であるか、path の最後が "¥"、":" でないときは、path の後に "¥" を加えます。

戻 り 値 戻り値はありません。

参 照 関 数 strmf

strncpy, strncat, strncmp, strnset

レベル 2

書 式 #include <string.h>

```
char *strncat(string1, string2, n);
char *string1; /* 文字の追加が行われる文字列へのポインタ */
char *string2; /* 追加文字を含む文字列のポインタ */
size_t n; /* 追加する文字数 */

int strncmp(string1, string2, n);
char *string1, *string2; /* 比較が行われる2つの文字列を指すポインタ */
size_t n; /* 比較する文字数 */

char *strncpy(string1, string2, n);
char *string1; /* 複写先文字列を指すポインタ */
char *string2; /* 複写元文字列を指すポインタ */
size_t n; /* 複写する文字数 */

char *strnset(string, c, n);
char *string; /* 文字の置き換えが行われる文字列へのポインタ */
char c; /* 置き換え文字 */
size_t n; /* 置き換えを行う文字数 */
```

機 能 末尾がヌル文字'¥0'の文字列に対して、次の操作を行います。

【strncat】

string1 の後ろに string2 の先頭から n 文字を追加し、末尾にヌル文字を付加します。

【strncmp】

string1 と string2 の先頭から n 文字を文字コードの順序で比較します。
大文字と小文字は区別されます。

【strncpy】

文字列 string2 の先頭から n 文字を string1 に複写します。

string2 の長さが n よりも小さい場合は、string1 の長さが n になるまで末尾にヌル文字を付加します。

string2 の長さが n よりも大きい場合は、ヌル文字は付加されません。

【strnset】

string の最初から n 文字分を文字 c に置き換えます。

string の長さが n よりも小さい場合はすべての文字が c に置き換えられます。

戻り値

以下の戻り値を返します。

【strncat】

新しい文字列を指すポインタを返します。

【strncmp】

戻り値とその内容は次の通りです。

戻り値	内 容
負	string1 は string2 より小さい
0	string1 と string2 は等しい
正	string1 は string2 より大きい

【strncpy】

string1 を指すポインタを返します。

【strnset】

新しい文字列を指すポインタを返します。

プログラム例

```

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

static char  p[256];

main()
{
    printf("p          -> %s\n",
           strncpy(p, "Sharp", 5));
    printf("p          -> %s\n",
           strncat(p, "x68000", 6));
    printf("strncmp(p, ¥\"sharp¥\"    -> %d\n",
           strncmp(p, "sharp", 5));
    printf("strnset(p, 'x', 5)      -> %s\n",
           strnset(p, 'x', 5));
}

```

strpbrk

レベル 2

書 式 #include <string.h>

```
char *strpbrk(string1, string2);  
char *string1; /* 検索が行われる文字列へのポインタ */  
char *string2; /* 検索する文字セットへのポインタ */
```

機 能 文字列 string1 の中で文字列 string2 に含まれている文字を検索し、最初に出現する位置を求めます。
ただし、ヌル文字'¥0'は検索できません。

戻 り 値 出現位置を指すポインタを返します。
該当する文字が見つからない場合には NULL を返します。

参 照 関 数 strchr, strrchr

プログラム例

```
#include <stdio.h>  
#include <string.h>  
  
static char buf[] = "Sharp x68000";  
static char set[] = "0123456789";  
  
main()  
{  
    printf("buf = %s¥n", buf);  
    printf("set = %s¥n", set);  
    printf("result = %s¥n", strpbrk(buf, set));  
}
```

strchr

レベル 2

書 式 #include <string.h>

```
char *strchr(string, c);
char *string; /* 検索が行われる文字列のポインタ */
int c; /* 検索する文字 */
```

機 能 文字列 string の中で文字 c を検索し、最後に出現する位置を求めます。
ヌル文字'¥0'も検索できます。

戻 り 値 出現位置を指すポインタを返します。
発見できなかった場合は NULL を返します。

参 照 関 数 strchr, strpbrk

プログラム例

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

main()
{
    char *str;
    int c;

    str = "This is a pen.";
    c = 'i';
    printf("str ->%s¥n", str);
    printf("c ->%c¥n", c);
    printf("strchr() ->%s¥n", strchr(str, c));
    printf("strrchr() ->%s¥n", strrchr(str, c));
}
```

strrev

レベル 2

書 式 #include <string.h>

```
char *strrev(string);  
char *string; /* 並び順の置き換えを行う文字列へのポインタ */
```

機 能 指定した文字列 string の文字列の並び順を逆にします。
ただしヌル文字'¥0'の位置は変化しません。

戻 り 値 並べ替えられた文字列を指すポインタを返します。

参 照 関 数 strcpy, strset

プログラム例

```
#include <stdio.h>  
#include <string.h>  
  
static char str[] = "This is a string."  
  
main()  
{  
    printf("normal ->%s¥n", str);  
    printf("reverse->%s¥n", strrev(str));  
}
```


strset

レベル 2

書 式 #include <string.h>

```
char *strset(string, c) ;  
char *string; /* 文字の置き換えが行われる文字列へのポインタ */  
char c; /* 置き換え文字 */
```

機 能 指定した文字列 string のすべての文字を、文字 c に置き換えます。
ただし末尾のヌル文字'¥0'は変化しません。

戻 り 値 変換された文字列を指すポインタを返します。

参照関数 strnset

プログラム例

```
#include <stdio.h>  
#include <string.h>  
  
static char str[] = "This is a string.";  
  
main()  
{  
    printf("normal ->%s¥n", str);  
    printf("convert->%s¥n", strset(str, '@'));  
}
```

strsfn

レベル 2

書 式 #include <string.h>

```
void strsfn(fname, drv, path, node, ext) ;  
char *fname, *drv, *path, *node, *ext ;
```

機 能 ファイル名 fname を分解して、それぞれの引数に格納します。
引数との対応は次の通りです。

drv : ドライブ名
path : パス名
node : 主ファイル名
ext : 拡張子

戻 り 値 戻り値はありません。

参 照 関 数 strmf

strspn

レベル 2

書 式 #include <string.h>

```
size_t strspn(string1, string2);
char *string1; /* 検索が行われる文字列へのポインタ */
char *string2; /* 文字セットへのポインタ */
```

機 能 string1 の先頭から、string2 に含まれる文字が何文字続いているかを調べます。

戻 り 値 何文字続いていたかをバイト数で返します。

最初から string2 に含まれる文字がなければ 0 を返します。

すべてが string2 に含まれている文字ならば string1 の全長（ヌル文字は含めない）を返します。

参 照 関 数 strchrspn

プログラム例

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

char str[] = "sharp Computer X6800.";
char set[] = "_ abcdefghijklmnopqrstuvwxyz";

main()
{
    printf("string      ->%s\n", str);
    printf("sufix length ->%d\n", strspn(str, set));
}
```

strsrt

レベル 2

書 式 #include <string.h>

```
void strsrt(s, n);  
char *s[]; /* 文字列へのポインタ配列 */  
int n; /* 配列の要素数 */
```

機 能 sが指す文字列の配列を ASCII コード順にソートします。
n には配列の要素数を指定します。

配列のサイズが大きいときには tqsort 関数を使用したほうが効率的です。

また、getfnl、strbpl 関数と組み合わせて使うと便利です。

戻 り 値 戻り値はありません。

プログラム例

```
#include <stdio.h>  
#include <string.h>  
  
main(int argc, char *argv[])  
{  
    int i;  
  
    for (i = 0; i < argc; i++)  
        printf("%s\n", argv[i]);  
  
    printf("sorting...\n");  
    strsrt(argv, argc);  
  
    for (i = 0; i < argc; i++)  
        printf("%s\n", argv[i]);  
}
```

strstr

レベル 2

書 式 #include <string.h>

```
char *strstr(string1, string2);
char *string1; /* 検索対象文字列へのポインタ */
char *string2; /* 検索文字列へのポインタ */
```

機 能 string1 で指定された検索対象文字列の中から、string2 で指定された検索文字列に最初に一致するところのポインタを返します。

戻 り 値 検索文字列が見つかった場合はそのポインタを返します。
見つからなかった場合は NULL を返します。

プログラム例

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

main()
{
    char *str;
    char *pat;
    char *p;

    str = "This is an apple, and this is a banana.";
    pat = "an";

    printf("%40s\n", str);
    while ((p = strstr(str, pat)) != NULL) {
        printf("%40s\n", p);
        str = p + 1;
    }
}
```

strtod

レベル 2

書 式 #include <stdlib.h>

```
double strtod(str, endptr);  
char *str; /* 変換する文字列 */  
char **endptr; /* 変換終了位置を返すためのポインタ */
```

機 能 strtod は文字列 str を倍精度(double 型)の浮動小数点値に変換します。
変換できる文字列の形式を以下に示します。

[空白(またはタブ文字)] [符号] [数字] [. 数字] [E(またはe)[符号] 数字]

[空白(またはタブ文字)] は無視されます。

[符号] は"+"または"-"です。

"+"のときは省略できます。

[数字] は 1 桁以上の 10 進数です。

これがないときは、[. 数字] に少なくとも 1 桁の数字が必要です。

[. 数字] は小数点以下の数字です。

また、-123E-12 のように指数表示することもできます。

strtod は、double の数値として解釈できない文字が現れると、変換をその時点で打ち切ります。

endptr が NULL でなければ、endptr には残りの文字の位置を設定します。

すべて文字が変換できたならば文字列終端のヌル文字'\0'の位置を、1 文字も変換出来なかったならば、文字列の先頭の位置を設定します。

戻 り 値 変換した double の値を返します。

オーバーフローが発生した場合は 0 を返します。

参 照 関 数 strtol, strtoul, atof

プログラム例

```

#include      <stdio.h>
#include      <stdlib.h>

#define STR_A  "1.49597870e11"
#define STR_G  "6.672e-11"

main()
{
    double a;
    double g;

    a = strtod(STR_A, NULL);
    printf("%s -> %g\n", STR_A, a);
    g = strtod(STR_G, NULL);
    printf("%s -> %g\n", STR_G, g);
}

```

strtok

レベル 2

書 式 `#include <string.h>`

```
char *strtok(string1, string2);  
char *string1; /* トークンを含む文字列へのポインタ */  
char *string2; /* 区切り文字を含む文字列へのポインタ */
```

機 能

string1 から、文字列 string2 の各要素の文字を区切り文字として区切られる文字列をトークンとして切り出します。

同一文字列から 2 回以上トークンを切り出す場合、2 回目以降の strtok の string1 には NULL を指定します。

このようにすると、次のトークンを検索することができます。

また区切り文字は strtok 関数を呼び出すたびに変わるので、string2 の値は何でもかまいません。

戻 り 値

切り出したトークンを指すポインタを返します。

トークンはすべてヌル文字'¥0'で終了します。

トークンを切り出せなかった場合は NULL を返します。

参 照 関 数

strcspn, strspn

プログラム例

```
#include <stdio.h>  
#include <string.h>  
  
static char *str = "data1,data2,data3|abc;def:key word";  
static char *dlim = "¥n¥t ,;:|";  
  
main()  
{  
    char *tp;  
  
    tp = strtok(str, dlim);  
    printf("token = %s¥n", tp);  
    while ((tp = strtok(NULL, dlim)) != NULL)  
        printf("token = %s¥n", tp);  
}
```


strtol

レベル 2

書 式

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

```
long strtol(str, endptr, radix);  
char *str; /* 変換する文字列 */  
char **endptr; /* 変換終了位置を返すためのポインタ */  
int radix; /* 指数指定 */
```

機 能

strtol は文字列 str を進数 radix にしたがって、符号付き整数に変換します。

radix は、文字列が何進数で表現されているかを指定します。

指定可能な範囲は、0 か 2~36 です。

変換できる文字列の形式は、radix の指定により異なります。

それぞれの場合に分けて、以下に示します。

● radix が 0 のとき

[空白(またはタブ文字)] [符号] [0 で始まらない 10 進数字]

[空白(またはタブ文字)] [符号] [0 で始まる 8 進数字]

[空白(またはタブ文字)] [符号] [0B(または 0b) で始まる 2 進数字]

[空白(またはタブ文字)] [符号] [0X(または 0x) で始まる 16 進数字]

[空白(またはタブ文字)] は無視されます。

[符号] は "+" または "-" です。

"+" のときは省略されます。

[10 進数字] は、0~9 までの文字です。

[8 進数字] は、0~7 までの文字です。

[2 進数字] は、0 か 1 の文字です。

[16 進数字] は、0~9、A~F(または a~f) の文字です。

● radix が 2 のとき

[空白(またはタブ文字)] [符号] [0B(または 0b) で始まる 2 進数字]

[空白(またはタブ文字)] は無視されます。

[符号] は "+" または "-" です。"+" のときは省略されます。

[0B(または 0b)] は、無くてもかまいません。

[2 進数字] は、0 か 1 の文字です。

● radix が 16 のとき

[空白(またはタブ文字)] [符号] [0X(または 0x) で始まる 16 進数字]

[空白(またはタブ文字)] は無視されます。

[符号] は "+" または "-" です。"+" のときは省略されます。

[0X(または 0x)] は、無くてもかまいません。

[16進数字] は、0~9、A~F(または a~f)の文字です。

- radix が 0、2、16 以外のとき

[空白(またはタブ文字)] [符号] [指定数字]

[空白(またはタブ文字)] は無視されます。

[符号] は”+”または”-”です。”+”のときは省略されます。

[指定進数字] は、0~9、A~Z(または a~z)の文字の最初から radix 個の文字です。

strtol は、指定の進数字の範囲外の文字が現れると、変換をその時点で打ち切ります。

endptr が NULL でなければ、endptr には残りの文字の位置を設定します。

すべての文字が変換できたならば文字列終端のヌル文字'¥0'位置を、一文字も変換出来なかったならば文字列の先頭の位置を設定します。

radix が 0 のときは、010 は 8 進数の 10、0B10 は 2 進数の 10 として変換します。

戻り値

変換した long の値を返します。

オーバーフローが発生した場合、負の方向でのオーバーフローならば LONG_MIN を、正の方向でのオーバーフローならば LONG_MAX を返し、それとともに errno に ERANGE を設定します。

1 文字も変換できなかった場合は 0 を返します。

参照関数

strtoul, strtod

プログラム例

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

main()
{
    char    *rp;
    long    l;

    l = strtol("123", &rp, 10);
    printf("long = %ld, p->%s¥n", l, rp);
    l = strtol("123", &rp, 8);
    printf("long = %ld, p->%s¥n", l, rp);
    l = strtol("123", &rp, 2);
    printf("long = %ld, p->%s¥n", l, rp);
}
```

strtoul

レベル 2

書 式 #include <stdlib.h>

```
unsigned long strtoul(str, endptr, radix);
char *str; /* 変換する文字列 */
char **endptr; /* 変換終了位置を返すためのポインタ */
int radix; /* 指数指定 */
```

機 能 strtoul は文字列 str を進数 radix にしたがって符号無し整数に変換します。
radix は、文字列が何進数で表現されているかを指定します。
指定可能な範囲は、0 か 2~36 です。
変換できる文字列の形式は、radix の指定により異なります。
それぞれの場合に分けて、以下に示します。

● radix が 0 のとき

[空白(またはタブ文字)] [符号] [0 で始まらない 10 進数字]
[空白(またはタブ文字)] [符号] [0 で始まる 8 進数字]
[空白(またはタブ文字)] [符号] [0B(または 0b)で始まる 2 進数字]
[空白(またはタブ文字)] [符号] [0X(または 0x)で始まる 16 進数字]
[空白(またはタブ文字)] は無視されます。
[10 進数字] は、0~9 までの文字です。
[8 進数字] は、0~7 までの文字です。
[2 進数字] は、0 か 1 の文字です。
[16 進数字] は、0~9、A~F(または a~f)の文字です。

● radix が 2 のとき

[空白(またはタブ文字)] [符号] [0B(または 0b)で始まる 2 進数字]
[空白(またはタブ文字)] は無視されます。
[0B(または 0b)] は、無くてもかまいません。
[2 進数字] は、0 か 1 の文字です。

● radix が 16 のとき

[空白(またはタブ文字)] [符号] [0X(または 0x)で始まる 16 進数字]
[空白(またはタブ文字)] は無視されます。
[0X(または 0x)] は、無くてもかまいません。
[16 進数字] は、0~9、A~F(または a~f)の文字です。

- radix が 0、2、16 以外のとき

[空白(またはタブ文字)] [符号] [指定数字]

[空白(またはタブ文字)] は無視されます。

[指定進数字] は、0~9、A~Z(または a~z) の文字の最初から radix 個の文字です。

strtoul は、指定の進数字の範囲外の文字が現れると、その時点で変換を打ち切ります。

endptr が NULL でなければ、endptr には残りの文字の位置を設定します。すべての文字が変換できたならば文字列終端のヌル文字 '\0' 位置を、1 文字も変換出来なかったならば文字列の先頭の位置を設定します。

radix が 0 のときは、010 は 8 進数の 10、0B10 は 2 進数の 10 として変換します。

戻り値

変換した unsigned long の値を返します。

オーバーフローが発生した場合は ULONG_MAX を返し、それとともに errno に ERANGE を設定します。

1 文字も変換できなかった場合は 0 を返します。

参照関数

strtoul, strtod

プログラム例

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

main()
{
    char *rp;
    unsigned long ul;

    ul = strtoul("123", &rp, 10);
    printf("long = %lu, p->%s\n", ul, rp);
    ul = strtoul("123", &rp, 8);
    printf("long = %lu, p->%s\n", ul, rp);
    ul = strtoul("123", &rp, 2);
    printf("long = %lu, p->%s\n", ul, rp);
}
```

strupr

レベル 2

書 式 #include <string.h>

```
char *strupr(string);
char *string; /* 変換が行われる文字列へのポインタ /*
```

機 能 文字列 string のすべての英小文字を、英大文字に置き換えます。
他の文字は変化しません。

戻 り 値 string を指すポインタを返します。

参 照 関 数 strtolwr

プログラム例

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

main()
{
    char *p;

    p = "this is lower";
    printf("original ->%s\n", p);
    printf("converted ->%s\n", strupr(p));

    p = "THIS IS UPPER";
    printf("original ->%s\n", p);
    printf("converted ->%s\n", strupr(p));

    p = "This one is $1200.";
    printf("original ->%s\n", p);
    printf("converted ->%s\n", strupr(p));
}
```

swab

レベル 2

書 式 #include <stdlib.h>

```
void swab(source, destination, n) ;  
char *source; /* スワップを行うデータへのポインタ */  
char *destination; /* スワップ後のデータ格納領域へのポインタ */  
int n; /* スワップを行うデータのバイト数 */
```

機 能

source から n バイトコピーし、隣接するバイトの並び、すなわち偶数アドレスのバイトと奇数アドレスのバイトを入れ替え、その結果を destination にストアします。

n は偶数値でなければなりません。

swab はふつう、異なったバイト並びをもったマシン語へ変換するために使用します。

戻 り 値

戻り値はありません。

参 照 関 数

swaw

プログラム例

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

#define SIZE 256

static char src[SIZE];
static char des[SIZE];

main()
{
    void setsub(char *);
    void putsub(char *);

    setsub(src);
    putsub(src);
    swab(src, des, SIZE);
    putsub(des);
}

static void setsub(char *p)
{
    int i;

    for (i = 0; i < SIZE; i++)
        *p++ = i;
}

static void putsub(char *p)
{
    int i;

    for (i = 0; i < SIZE; i++) {
        printf("%02x ", (*p++ & 0xff));
        if (i % 16 == 15)
            putchar('\n');
    }
    putchar('\n');
}
```

swaw

レベル 2

書 式 #include <stdlib.h>

```
viod swaw(src, dst, num) ;  
int *src; /* スワップを行うデータへのポインタ */  
int *dst; /* スワップ後のデータ格納領域へのポインタ */  
int num; /* スワップを行うデータのバイト数 */
```

機 能 src から num で指定したバイト分コピーして、int 型と同じサイズ (4 バイト) を 1 組とするデータのバイトの並びを逆にします。
結果は、dst に格納されます。
num は 4 の倍数でなければなりません。

戻 り 値 戻り値はありません。

参 照 関 数 swab

swmem

レベル 2

書 式 `#include <string.h>`

```
void swmem(p, q, size) ;
void *p; /* 領域 1 へのポインタ */
void *q; /* 領域 2 へのポインタ */
size_t size; /* 入れ替えるバイト数 */
```

機 能 p が指す領域の size バイトと、q が指す領域の size バイトの内容を入れ替えます。

戻 り 値 戻り値はありません。

プログラム例

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

main()
{
    static char    p[] = "Sharp";
    static char    q[] = "x68000";

    printf("p = %s\n", p);
    printf("q = %s\n", q);

    swmem(p, q, 4);

    printf("p = %s\n", p);
    printf("q = %s\n", q);
}
```

参 照

symbol

レベル 3

書 式 #include <graph.h>

```
void symbol(x, y, st, h, v, mo, p, an) ;  
int x; /* 始点 X 座標 */  
int y; /* 始点 Y 座標 */  
char *st; /* 文字列を指すポインタ */  
char h; /* 横倍率 */  
char v; /* 縦倍率 */  
int mo; /* フォントの種類 */  
int p; /* パレットコード */  
int an; /* 回転 */
```

機 能

グラフィック画面上に文字列を表示します。

クリッピング領域は、window 関数で指定した範囲になります。

x には始点 X 座標を指定します (−32768~32767)。

y には始点 Y 座標を指定します (−32768~32767)。

p にはパレットコードを指定します。

mo にはフォントの種類を指定します。

0 : 6×12

1 : 8×16

2 : 12×24

an には回転数を指定します (0~3)。

1 単位は 90°です。

st には表示文字列を指すポインタを指定します。

h には横倍率を指定します。

v には縦倍率を指定します。

戻 り 値

戻り値はありません。

2424

system

レベル 2

書 式 #include <stdlib.h>

```
int system(string) ;  
char *string; /* Human68k コマンドの文字列を指すポインタ */
```

機 能 文字列 string を Human68k のコマンドとしてコマンドプロセッサ (COMMAND. X) にわたします。

system は、COMMAND. X を探すために path の環境変数を参照します。

戻 り 値 正の数 (0 も) で正常終了、-1 でエラー。
正の数の場合、戻り値は起動した子プロセス (COMMAND. X) の終了コード。

参 照 関 数 execl, execl, execlp, execv, execve, execvp, exit, _exit, spawnl, spawnle, spawnlp, spawnv, spawnve, spawnvp

プログラム例

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

```
main(int argc, char *argv[])  
{  
    char *str;  
  
    str = (argc == 1) ? "date" : argv[1];  
  
    if (system(str) < 0) {  
        printf("error\n");  
        exit(1);  
    }  
    exit(0);  
}
```

tan, tanh

レベル 2

書 式 #include <math.h>

double tan(x) ;

double tanh(x) ;

double x ;

機 能 tan は、x のタンジェントを、tanh は x のハイパボリック (双曲線) タンジェントを計算します。

戻 り 値 tan は x のタンジェント値を返します。
tanh はハイパボリックタンジェント値を返します。

参 照 関 数 acos, asin, atan, atan2, cos, cosh, sin, sinh

プログラム例

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

main()
{
    double x;
    int i;

    for (i = -32; i <= 32; i++) {
        x = (double)i / 10.0;
        printf("tan(%.1f) = %g\n", x, tan(x));
    }
    for (i = -32; i <= 32; i++) {
        x = (double)i / 10.0;
        printf("tanh(%.1f) = %g\n", x, tanh(x));
    }
}
```

tell

レベル 2

書 式 #include <io.h>

```
long tell(handle) ;  
int handle; /* 該当ファイルのファイルハンドル */
```

機 能 ファイルハンドル handle のファイルポインタの現在位置を調べます。

戻 り 値 現在の位置を先頭からのバイト数で返します。
ファイルハンドルの引数が無効の場合は-1を返し、errnoにEBADFがセットされます。
シーク不可能なデバイス (RS-232C やプリンタなど) を指定した場合、戻り値は不定となります。

参 照 関 数 ftell, lseek

プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
#include <fcntl.h>  
#include <io.h>  
#include <stat.h>  
  
#define LEN 100L  
  
main(int argc, char *argv[])  
{  
    char    buf[LEN];  
    char    *file;  
    int     fn;  
  
    file = (argc > 1) ? argv[1] : "file";  
    if ((fn = open(file, O_RDONLY|O_BINARY)) == -1) {  
        perror(file);  
        exit(1);  
    }  
    if (lseek(fn, -LEN, SEEK_END) == -1L) {  
        perror("seek error");  
        exit(1);  
    }  
    printf("pos = %ld\n", tell(fn));  
    close(fn);  
    exit(0);  
}
```

tempfile

レベル 2

書 式 #include <stdio.h>

```
FILE *tempfile(dir, prefix);
char *dir; /* 作成するファイルのパス名の部分 */
char *prefix; /* 作成するファイルの基本部分 */
```

機 能

tempfile は、テンポラリファイルをバイナリモードで更新用("w+b")にオープンします。

作成されるテンポラリファイルは、クローズしたり、プログラムが exit 関数によって正常終了した時点で、自動的に削除されます。

tempfile は、オープンされたファイルが自動的に削除されることを除けば、fopen(tempnam(dir, prefix), "w+b") と同じ動作をします。

dir や prefix については、tempnam 関数を参照して下さい。

tempfile は、ファイル名を内部の静的な領域で作成し、そのファイル名を用いてファイルをオープンします。

この静的な領域は tempnam や tmpnam 関数などでも使用するため、tempfile を呼び出すと、それらの関数が作成したファイル名を消してしまうので注意して下さい。

戻 り 値

オープンに成功すると、オープンしたストリームのポインタを返します。
オープンに失敗すると、ヌルポインタを返します。

参 照 関 数

tmpfile, tempnam, tmpnam

プログラム例

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

main()
{
    FILE *fp;

    if ((fp = tempfile("b:", "sample.tmp")) == NULL)
        fprintf(stderr, "can't open temp file\n");
    exit(0);
}
```

tempnam

レベル 2

書 式 #include <stdio. h>

```
char *tempnam(dir, prefix) ;  
char *dir; /* 作成するファイル名のパス名の部分 */  
char *prefix; /* 作成するファイル名の基本部分 */
```

機 能

tempnam は、既存のファイルとは異なるファイル名を作成します。

作成されたファイル名は、一時的に使用するテンポラリファイルの名前として使用することができます。

tempnam は、最大 TMP_MAX 回まで異なるファイル名を作成します。

dir には、パス名を指定します。

作成されるファイル名のパス名は dir になります。

prefix は、ファイル名の基本部分を指定します。

作成されるファイル名の主ファイル名は、prefix に付加した文字列になります。

prefix に拡張子が含まれていると、作成されるファイル名の拡張子はそれになります。

prefix に拡張子が含まれていないと、拡張子は“\$\$\$”になります。

dir が、カレントドライブ、カレントディレクトリを参照する(ドライブ名が含まれていない、またはパス名が含まれていない、相対パスを使用している場合など)ときは、次のことに注意してください。

作成した名前前でファイルをオープンする前に、カレントドライブカレントディレクトリを変更すると、作成したファイル名は意味をなさなくなってしまう。

tempnam は、ファイル名を内部の静的な領域に作成し、そのポインタを返します。

そのため、再度 tempnam や tmpnam などと呼び出すと、上書きして古いファイル名を消してしまいます。

この関数によって作成されたファイル名で普通に (fopen や open 関数などで) ファイルを作成した場合、作成されたファイルは、自動的に消されることはありません。

プログラマの責任によって削除してください。

戻 り 値

作成に成功すると、作成した文字列のポインタを返します。

作成に失敗すると、NULL を返します。

参 照 関 数

tmpfile, tempfile, tmpnam

プログラム例

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

```
main()
```

```
{
```

```
    char *name;
    FILE *fp;
```

```
    if ((name = tmpnam("B:", "sample.tmp")) == NULL)
        fprintf(stderr, "can't create temp file\n");
```

```
    if ((fp = fopen(name, "w")) == NULL) {
        fprintf(stderr, "can't open temp file\n");
        exit(1);
    }
```

```
    else {
```

```
        fprintf(fp, "hello\n");
        system("dir %s");
        system("dir");
        (void)fclose(fp);
        exit(0);
    }
```

```
}
```

time

レベル 2

書 式 `#include <time.h>`

```
time_t time(tp) ;  
time_t *tp;
```

機 能 システムクロックが、グリニッジ標準時 (GMT) 1970 年 1 月 1 日 0 時 0 分 0 秒から経過した秒数を求めます。
tp が NULL でなければ秒数は、tp にも格納されます。

戻 り 値 経過秒数を返します。

参 照 関 数 `asctime, ftime, gmtime, localtime, utime, ctime`

tmasc

レベル 2

書 式 #include <io.h>

```
char *tmasc(time);
int time; /* 時刻情報 */
```

機 能

time で指定された時刻情報を文字列に変換し、その文字列へのポインタを返します。

time の指定は以下の通りです。

ビット 15~11 : 時(0~23)

ビット 10~5 : 分(0~59)

ビット 4~0 : 秒(0~59/2)

time の形式は tmcnv 関数の戻り値と同じです。

time より変換・作成される文字列は、内部の静的なバッファに格納されるため、この関数を呼び出すと以前の内容は失われることに注意してください。

戻 り 値

正常終了ならば、内部の静的バッファに作成・格納された文字列へのポインタを返します。

返される文字列は、時を HH、分を MM、秒を SS とすると、"HH : MM : SS" という形式となります。

エラーならば NULL を返します。

参 照 関 数

tmcnv

プログラム例

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <stat.h>

main()
{
    char *p;
    int fn;
    int datetime;

    fn = open("¥¥CONFIG.SYS", O_RDWR);
    datetime = getft(fn);

    if ((p = dtasc(datetime >> 16)) != NULL)
        printf("日付: %s¥n", p);
    if ((p = tmasc(datetime & 0xffff)) != NULL)
        printf("時刻: %s¥n", p);
    close(fn);
}
```

tmcnv

レベル 2

書 式 #include <io. h>

```
int tmcnv(timestr) ;
char *timestr; /* 変換したい時刻文字列 */
```

機 能 timestr で指定された文字列を時刻情報に変換して、その値を返します。
timestr の指す文字列は、時を HH、分を MM、秒を SS とすると、"HH : MM : SS" という形式で指定します。
timestr は、tmasc 関数の戻り値と同じです。

戻 り 値 正常終了ならば、以下に示すフォーマットで時刻情報を返します。

```
ビット 15~11 : 時(0~23)
ビット 10~5  : 分(0~59)
ビット 4~0   : 秒(0~59/2)
```

エラーならば負の数を返します。

参 照 関 数 tmasc

プログラム例

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <stat.h>
```

```
main()
{
    int fn;
    int datetime;
    int date;
    int time;

    if ((date = dtcnv("90-07-14")) < 0)
        printf("エラー\n");
    if ((time = tmcnv("12:30:00")) < 0)
        printf("エラー\n");

    datetime = ((date << 16) | time);

    fn = open("%$CONFIG.SYS", O_RDWR);
    chgft(fn, datetime);
    close(fn);
}
```

tmpfile

レベル 2

書 式 #include <stdio.h>

FILE *tmpfile();

機 能 tmpfile は、テンポラリファイルをバイナリモードで更新用("w+b")にオープンします。

作成されるファイルは、クローズしたり、プログラムが exit 関数によって正常終了した時点で自動的に消去されます。

tmpfile は、オープンされたファイルが自動的に消去されることを除けば、fopen(tmpnam(NULL), "w+b") と同じ動作をします。

tmpfile は、ファイル名を内部の静的な領域に作成し、そのファイル名を用いてファイルをオープンします。

この静的な領域は tmpnam や tmpnam 関数などでも使用するため、tmpfile を呼び出すと、それらの関数が作成したファイル名を消してしまうので注意してください。

戻 り 値 オープンに成功すると、オープンしたストリームのポインタを返します。

オープンに失敗すると、NULL を返します。

参 照 関 数 tmpfile, tmpnam, tmpnam

プログラム例

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

main()
{
    FILE *fp1;
    FILE *fp2;

    if ((fp1 = tmpfile()) == NULL)
        fprintf(stderr, "can't open temp file\n");
    if ((fp2 = tmpfile()) == NULL)
        fprintf(stderr, "can't open temp file\n");

    fprintf(fp1, "hello");
    fprintf(fp2, "hello");
    system("dir %f");
    system("dir");

    (void)fclose(fp2);
    (void)fclose(fp1);
    exit(0);
}
```

tmpnam

レベル 2

書 式 #include <stdio.h>

```
char *tmpnam(str);  
char *str; /* 作成したファイル名を格納する領域へのポインタ */
```

機 能

tmpnam は、既存ファイルとは異なるファイル名を作成します。

作成されたファイル名は、一時的に使用するテンポラリファイルの名前として使用することができます。

tmpnam は、最大 TMP_MAX 回まで異なるファイル名を作成します。

str は、作成したファイル名を格納する領域へのポインタか、NULL を指定します。

領域を指定した場合、その領域は最低でも L_tmpnam 個の文字が格納できなければなりません。

NULL を指定した場合、ファイル名は内部の静的な領域に作成されます。

そのため、再度 tmpnam や tmpnam などと呼び出すと、上書きして古いファイル名を消してしまいますので注意してください。

この関数が作成するファイル名は、カレントドライブ、カレントディレクトリを変更すると、意味をなさなくなってしまうので注意してください。

この関数によって作成されたファイル名で普通に (fopen や open 関数などで) ファイルを作成した場合、作成されたファイルは、自動的に消されることはありません。

プログラマの責任によって削除してください。

戻 り 値

作成に成功したとき、str が NULL の場合、内部の静的な領域のポインタを返し、str が NULL ではない場合、str を返します。

作成に失敗すると、NULL を返します。

参 照 関 数

tempnam, tempfile, tmpfile

プログラム例

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

main()
{
    char      buf[L_tmpnam];
    char      *name;
    FILE      *fp;

    if ((name = tmpnam(buf)) == NULL)
        fprintf(stderr, "can't create temp file\n");
    if ((fp = fopen(name, "w")) == NULL)
        fprintf(stderr, "can't open temp file\n");

    fprintf(fp, "hello\n");
    system("dir %s");
    system("dir");

    (void)fclose(fp);
    exit(0);
}
```

toascii, tolower, toupper

_tolower, _toupper

レベル 2

書 式 #include <ctype.h>

```
int toascii(c) ;  
int tolower(c) ;  
int _tolower(c) ;  
int toupper(c) ;  
int _toupper(c) ;
```

```
int c; /* 文字 */
```

機 能 文字 c を次のように変換します。

【toascii】

文字 c の下位 7 ビット以外を 0 にして、ASCII 文字に変換します。

c が ASCII 文字を表している場合は変換しません。

【tolower】

文字 c が大文字の場合は小文字に変換します。

大文字以外の場合は変換しません。

【_tolower】

文字 c が大文字の場合は小文字に変換します。

大文字以外の場合は不定です。

【toupper】

文字 c が小文字の場合は大文字に変換します。

小文字以外の場合は変換しません。

【_toupper】

文字 c が小文字の場合は大文字に変換します。

小文字以外の場合は不定です。

戻 り 値 変換後の文字を返します。

エラー時の戻り値はありません。

参照関数

isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct,
isspace, isupper, isxdigit

プログラム例

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

main()
{
    int c;
    int c1, c2, c3, c4;

    printf(" c low upp _low _upp\n");
    for (c = 0x20; c < 0x7f; c++) {
        c1 = tolower(c);
        c2 = toupper(c);
        c3 = isupper(c) ? tolower(c) : c;
        c4 = islower(c) ? toupper(c) : c;
        printf("%02x: %c %c %c %c\n",
               c, c, c1, c2, c3, c4);
    }
}
```

注：以上はマクロ定義と関数版があります。マクロ定義には `ctype.h` を用い、関数版には `ctype.h` を用いてください。

書 式

```
#include <time.h>
```

```
void tzset( ) ;
```

```
int daylight;
```

```
time_t timezone;
```

```
char *tzname [2] ;
```

機 能

環境変数 TZ に設定されている値を利用して、次の変数に値を代入します。

timezone : グリニッジ標準時と地方時の時差秒数

tzname [0] : 3 文字のタイムゾーン文字列

tzname [1] : 3 文字の夏時間調節ゾーン文字列

daylight : tzname [1] が空文字列 (NULL) ならば 0

tzname [1] が空文字列 (NULL) でなければ夏時間調節秒数

これらの変数は、地方時を扱う関数においてグリニッジ標準時 (GMT) を地方時に変換するときに使います。

なお、プログラム起動時に自動的に tzset が呼び出されます。

環境変数 TZ には、以下のような値を設定しておきます。

JST-9 (日本の場合)

たとえば ABC5DEF1 ならば、タイムゾーン文字列は“ABC”、夏時間調節ゾーン文字列は“DEF”、時差は 5 時間、夏時間調節は 1 時間になります。

また、夏時間調節文字列の後の数字を省略するか、0 にすると 1 時間になります。

また、夏時間調節文字列がない場合は、夏時間はなしになります。

環境変数 TZ がない場合は、JST-9 (日本時間) になります。

環境変数 TZ の設定は、Human68K のコマンドラインで以下のように行うことができます。

```
A>SET TZ=GMT0 [J]
```

なお、時差は、グリニッジ標準時－地方時で求めます。

たとえば、グリニッジ標準時が 3 時のとき地方時が 13 時であれば、 $3-13=-10$ で、-10 時間になります。

uitoa

レベル 2

書 式 `#include <stdlib.h>`

`char *uitoa(val, str, rdx) ;`

`unsigned int val; /* 変換する数値 */`
`char *str; /* 文字列を格納する領域 */`
`int rdx; /* 基数 */`

機 能 数値 val を文字列 str に変換します。

rdx は val が何進数なのかを表し（基数という）、必ず 2 から 36 の範囲で指定します。

str は変換後の文字列が十分納まる大きさでなければなりません。

戻 り 値 str を指すポインタを返します。

プログラム例

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

main()
{
    char    buf[34];
    unsigned int    ui;

    for (ui = 0; ui < 60000U; ui += 1000) {
        printf("%4u : ", ui);
        printf("%7s ", uitoa(ui, buf, 2));
        printf("%3s ", uitoa(ui, buf, 8));
        printf("%2s ", uitoa(ui, buf, 10));
        printf("%2s ", uitoa(ui, buf, 16));
        printf("%n");
    }
}
```

ultoa

レベル 2

書 式 `#include <stdlib.h>`

```
char *ultoa(value, string, radix);
unsigned long value; /* 変換すべき整数値 */
char *string; /* 文字列の格納領域を指すポインタ */
int radix; /* 基数 */
```

機 能 数値 value をスル文字'¥0'で終了する文字列に変換し、string に格納します。
 オーバーフローのチェックは行われません。
 引数 radix は、value の基数で、 $2 \leq \text{radix} \leq 36$ の整数です。
 文字列 string を格納するためには、最大 33 バイトの領域が必要です。

戻 り 値 string を指すポインタを返します。

参 照 関 数 itoa, ltoa

プログラム例

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

main()
{
    char buf[34];
    unsigned long ul;

    for (ul = 0; ul < 1000000UL; ul += 10000UL) {
        printf("10lu : ", ul);
        printf("%20s ", ultoa(ul, buf, 2));
        printf("%8s ", ultoa(ul, buf, 8));
        printf("%7s ", ultoa(ul, buf, 10));
        printf("%6s ", ultoa(ul, buf, 16));
        printf("¥n");
    }
}
```

書 式 #include <stdio. h>

```
int ungetc(c, stream) ;
int c; /* 返される文字 */
FILE *stream; /* FILE構造体へのポインタ */
```

機 能	入力ストリーム stream から、直前に読み込んだ文字 c を入力ストリームに戻し、ファイルポインタを 1 文字分戻します。
-----	---

streamはバッファリングされ、読み込みのためにオープンしていなければなりません。

なお、`ungetc`を使ってEOFを戻そうとしても無視されます。

また、stream から何も読み込んでいない場合や、c が戻すことができない場合は、エラーとなります。

入力ストリームに戻した文字は、fseek や rewind 関数を実行すると消されます。

戻り値	入力ストリームに戻した文字を返します。
-----	---------------------

エラーが発生した場合は EOF を返します。

参照関数	getc, petchar, putc, putchar
------	------------------------------

プログラム例

```

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

main()
{
    void      out_digit(FILE *);
    FILE      *fp;
    int       c;

    while ((c = getchar()) != EOF) {
        if (isdigit(c)) {
            ungetc(c, stdin);
            out_digit(stdin);
        }
    }
    exit(0);
}

static void  out_digit(FILE *fp)
{
    int       c;

    while ((c = getchar()) != EOF) {
        if (isdigit(c))
            putchar(c);
        else {
            ungetc(c, fp);
            break;
        }
    }
}

```

ungetch

レベル 2

書 式 #include <conio.h>

```
int ungetch(c) ;  
int c; /* 戻す文字 */
```

機 能 文字 *c* をコンソールに戻し、次の読み込み文字とします。
文字を読み込まないで2回以上連続して ungetch を実行するとエラーになります。

戻 り 値 戻した文字を返します。
エラーが発生した場合は EOF を返します。

参 照 関 数 cscanf, getch, getche

プログラム例

```
#include <ctype.h>  
#include <conio.h>  
  
#define MAX 64  
  
main()  
{  
    char buf[MAX];  
    int i;  
    int c;  
  
    while ((c = getche()), isspace(c))  
        ;  
    for (i = 0; i < MAX; i++) {  
        if (isspace(c))  
            break;  
        buf[i++] = c;  
        c = getche();  
    }  
    ungetch(c);  
    buf[i] = '\0';  
}
```


unlink, remove

レベル 2

書 式 #include <io.h>

```
int unlink(pathname) ;
char *pathname; /* 削除するファイル */
```

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

```
int remove(pathname) ;
char *pathname; /* 削除するファイル */
```

機 能 ファイル pathname を削除します。

unlink と remove は定義されているヘッダファイルが異なるだけで、機能、実行結果は同じです。

戻 り 値 正の数 (0 も) で正常終了、-1 でエラー。

通常は 0 が返りますが、特殊デバイスドライバが、対象の時 0 以外の正の数が返ります。

参 照 関 数 close

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>

main(int argc, char *argv[])
{
    if (argc == 1)
        puts("Usage: unlink filename");
    else {
        if (unlink(argv[1]) == -1) {
            printf("%s を削除できません。 %n", argv[1]);
            exit(1);
        }
    }
    exit(0);
}
```

using

レベル 3

書 式 #include <basic0. h>

```
char *using(s, fmt, . . .) ;  
char *s; /* 変換した文字列を指すポインタ */  
char *fmt; /* 書式制御文字列へのポインタ */
```

機 能 * fmt の内容により、スタック上のデータが double か string かを判断します。
数値はすべて double です。
なお、書式制御文字は次の通りです。

● 数

制御文字	機 能
#	桁
.	小数点
+ -	符号
**	*で埋める
¥¥	¥記号をつける
**¥	¥記号をつけ*埋める
^^^	小数点
,	3 桁のカンマ区切り

● 文字

制御文字	機 能
!	先頭の 1 文字
&	“&”で囲んだだけの文字数分 (“&”も含む) 文字列を出力する
@	エスケープ文字 これに続く 1 文字を文字として印刷する

戻 り 値 作成した文字列を指すポインタを返します。

utime

レベル 2

書 式 `#include <utime.h>`

```
int utime(pathname, times) ;
char *pathname; /* 該当ファイルのパス名 */
struct utimbuf *times; /* utimbuf 構造体を指すポインタ */
```

機 能

ファイル pathname の更新日時を times にセットします。

ファイルは書き込み可能でなければなりません。

times は utime.h で定義されている utimbuf 型の構造体を指していなければなりません。utimbuf 型構造体は以下の通りです。

```
struct utimbuf {
    time_t actime;
    time_t modtime;
};
```

変更日時はこの構造体の modtime 領域によってセットされます。

times が NULL の場合は、変更日時は現在の時間にセットされます。

戻 り 値

更新日時を変更できたときには 0 を返します。

エラーが発生した場合は -1 を返し、errno に次の値がセットされます。

errno の値	内 容
EACCES	pathname がディレクトリを指定、または読み込み専用の属性が付いたファイルを指定している
EMFILE	オープンしているファイルの数が多すぎる
ENOENT	ファイルまたはパス名が見つからない

参 照 関 数

asctime, ctime, fstat, ftime, gmtime, stat, time

プログラム例

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

main()
{
    if (utime("¥¥sales¥¥west", NULL) != 0)
        perror("変更できません。");
}
```

uwtoa, wtoa

レベル 2

書 式 #include <stdlib.h>

```
char *wtoa(wval, str, rdx);
char *uwtoa(uval, str, rdx);

short wval; /* 変換する数値 */
unsigned short uval; /* 変換する数値 */
char *str; /* 文字列格納配列へのポインタ */
int rdx; /* 基数 */
```

機 能

数値 wval, uval を文字列 str に変換します。

rdx は val が何進数なのかを表し (基数という)、必ず 2 から 36 の範囲で指定します。

str は変換後の文字列が十分に納まる大きさでなければなりません。

wtoa で、wval が負の数、rdx が 10 のときは、str の 1 文字目はマイナス (-) になります。

戻 り 値

str を指すポインタを返します。

プログラム例

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

main()
{
    char buf[34];
    short s;
    for (s = -100; s < 100; s += 10) {
        printf("%7ld: ", s);
        printf("%032s", wtoa(s, buf, 2));
        printf("%11s ", wtoa(s, buf, 8));
        printf("%7s ", wtoa(s, buf, 10));
        printf("%8s ", wtoa(s, buf, 16));
        printf("\n");
    }
}
```

**va_start, va_arg,
va_end****レベル 2****書 式** `#include <stdarg.h>`

```
viod va_start(param, lastarg) ;  
type va_arg(param, type) ;  
viod va_end(param) ;  
va_list param ;
```

機 能

`va_start`, `va_arg`, `va_end` は、可変個の引数の参照を実現するために用意されています。

`param` は、可変個の引数を参照するための変数です。

可変個の引数を参照する関数の先頭で、`va_list` 型の変数として宣言します。

`va_start` は、`param` が可変個の引数の先頭を指すように設定します。

`lastarg` は、その関数の固定の引数のうち、最後の引数の名前を設定します。

`va_arg` は、可変個の引数のうち、`param` が指す引数を `type` 型の値として取り出します。

そして、`param` が次の引数を指すように変更します。

`va_end` は、可変個の引数の参照を終了させ、`va_start`, `va_arg` を使用した関数から呼び出し元へ、正常に制御が戻るように手続きをします。

`va_start`, `va_arg`, `va_end` は、`stdarg.h` でマクロとして宣言されています。

戻 り 値

`va_start`, `va_end` は、戻り値がありません。

`va_arg` は、可変個の引数のうち、`param` が指す引数を、`type` 型の値として返します。

参 照 関 数

`vfprintf`, `vprintf`, `vsprintf`

プログラム例

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

main()
{
    char *addstr(char *, ...);
    char buf[256];
    char *ptr;

    ptr = addstr(buf, "abc", "123", "ABC", NULL);
    printf("%s\n", ptr);
    ptr = addstr(buf, "123", "ABC", "ABC", "XYZ", NULL);
    printf("%s\n", ptr);
}

static char *addstr(char *buf, ...)
{
    va_list ap;
    char *s;

    va_start(ap, buf);
    *buf = '\0';
    while ((s = va_arg(ap, char *)) != NULL)
        strcat(buf, s);
    va_end(ap);
    return (buf);
}
```

大 書

読 者

読 者

読 者

val

レベル 3

書 式 #include <basic.h>

```
double val(s) ;
```

```
char *s; /* 変換を行う文字列へのポインタ */
```

機 能

文字列を数値に変換します。
数値には、X-BASIC の表現 (&H など) を使うこともできます。

戻 り 値

変換後の数値を返します。

v_cut

レベル 3

書 式 #include <image. h>

```
void v_cut(tv_sw) ;  
char tv_sw; /* 制御スイッチ */
```

機 能 スーパーインポーズ表示時、テレビ画面の表示制御を行います。
tv_sw に指定できる内容は次の通りです。

- 0 : テレビ画面の表示
- 1 : テレビ画面の表示カット

また、次の関数の組み合わせにより、各種の画面を表示することができます。

v_cut (0) : vpage (1) …… スーパーインポーズ
v_cut (0) : vpage (0) …… テレビ画面
v_cut (1) : vpage (1) …… コンピュータ (グラフィック)

戻 り 値 戻り値はありません。

fprintf, vprintf, vsprintf, vfprintf

レベル 2

書 式 `#include <stdio.h>`

```
int fprintf(stream, format, argptr) ;
int vprintf(format, argptr) ;
int vsprintf(buffer, format, argptr) ;
FILE *stream; /* 出力するファイルストリーム */
char *format; /* 出力書式 */
char *buffer; /* 出力先文字列バッファ */
va_list argptr; /* 引数へのポインタ */
```

`#include <conio.h>`

```
int vfprintf (format, argptr) ;
char *format; /* 出力書式 */
va_list argptr; /* 引数へのポインタ */
```

機 能

`fprintf`, `vprintf`, `vsprintf`, `vfprintf` 関数は、それぞれ `fprintf`, `printf`, `sprintf`, `cprintf` とほぼ同じ機能を持つ関数です。

これらの違いは、出力したい引数を直接指定するのではなく、引数リストへのポインタを指定する点です。

各関数の詳細は、それぞれに対応する関数を参照してください。

動作については、そちらを参照して下さい。

引数リストへのポインタ `argptr` は、`va_start` 関数によって設定される可変個の引数を参照するための変数を指定します。

例えば、`printf` 関数を `vprintf` を用いて実現すると、以下のようになります。

```
int printf(format)
char *format;
{
    va_list argptr;
    int status;
    va_start(argptr, format) ;
    status = vprintf(format, argptr) ;
    va_end(argptr) ;
    return(status) ;
}
```

戻り値

出力した文字数を返します。

エラーの時は負の数を返します。

参照関数

fprintf, printf, sprintf, va_start, va_end, cprintf

プログラム例

```

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

main()
{
    int eprintf(char *);

    printf("これは標準出力です\n");
    eprintf("これは標準エラー出力です\n");
}

static int eprintf(char *format)
{
    va_list argptr;
    int status;
    va_start(argptr, format);
    status = vfprintf(stderr, format, argptr);
    va_end(argptr);
    return (status);
}

```

第

章

vpage

レベル 3

書 式 #include <graph. h>

```
void vpage(i) ;
char i; /* 表示ページの指定 */
```

機 能

グラフィック画面の各ページ (0~3) の表示をするかしないかを設定します。

i のビット 0~3 がページ 0~3 にそれぞれ対応し、1 (表示 ON) または 0 (表示 OFF) で表します。

また、i は次のような範囲の値になります。

スクリーンモード	i	ページ数
1024×1024 (16 色)	0, 1	1
512×512 (16 色)	0~15	4
512×512 (256 色)	0~3	2
512×512 (65536 色)	0, 1	1

戻 り 値

戻り値はありません。

wait

レベル 2

書 式 #include <process.h>

int wait();

機 能 現在、実行しているプログラムが直前に実行した子プロセスの終了コードを受け取ります。

戻 り 値 子プロセスの終了コードを返します。

戻り値	説明
0	子プロセスが正常終了したことを示す。
1	子プロセスが異常終了したことを示す。
2	子プロセスが正常終了したことを示す。
3	子プロセスが異常終了したことを示す。

備 考 戻り値は子プロセスの終了コード。

width

レベル 3

書 式 #include <basic0.h>

void width(n) ;

int n; /* 1 行の表示文字数 */

機 能

画面の 1 行の表示文字数を設定します。

n は 64 か 96 のどちらかを設定してください。

戻 り 値

戻り値はありません。

```

#include <basic0.h>
#include <graph.h>

main()
{
    int i;
    p_init();
    screen(2, 5, 1, 1);
    vplane();

    wipe();
    window(0, 0, 787, 611);
    line(0, 0, 787, 611, 15, 0x0000);

    for (i = 0; i < 3000; i++)
    {
        wipe();
        window(0, 0, 787, 611);
        line(0, 0, 787, 611, 15, 0x0000);
    }
    p_exit(0);
}

```

window

レベル 3

書 式 `#include <graph.h>`

```
void window (x1, y1, x2, y2) ;  
int x1; /* 始点 X 座標 */  
int y1; /* 始点 Y 座標 */  
int x2; /* 終点 X 座標 */  
int y2; /* 終点 Y 座標 */
```

機 能

グラフィック画面のクリッピング領域の座標を指定します。

x1 には始点 X 座標を指定します。

y1 には始点 Y 座標を指定します。

x2 には終点 X 座標を指定します。

y2 には終点 Y 座標を指定します。

上記の座標の指定可能範囲は、実画面サイズにより 0～511、または 0～1023 となります。

座標を指定するときは $x1 \leq x2$ 、 $y1 \leq y2$ となっていなければいけません。

戻 り 値 戻り値はありません。

参 照 関 数 `screen`

プログラム例

```
#include <basic0.h>  
#include <graph.h>  
  
main()  
{  
    int i;  
  
    b_init();  
    screen(2, 0, 1, 1);  
    vpage(1);  
  
    wipe();  
    window(0, 0, 767, 511);  
    line(0, 0, 767, 511, 15, 0xffff);  
  
    for (i = 0; i < 3000; i++)  
        ;  
  
    wipe();  
    window(0, 0, 200, 200);  
    line(0, 0, 767, 511, 15, 0xffff);  
  
    b_exit(0);  
}
```

wipe

レベル 3

書 式 #include <graph.h>

void wipe () ;

機 能 グラフィック画面をクリアします。

戻 り 値 戻り値はありません。

参 照 関 数 apage

プログラム例

```
#include <basic0.h>
#include <graph.h>

main()
{
    int i, j;

    b_init();
    screen(2, 0, 1, 1);
    vpage(1);

    fill(0, 0, 767, 511, 15);
    wipe();
    screen(1, 1, 1, 1);
    vpage(0);

    apage(0);
    fill(50, 50, 200, 200, 3);
    apage(1);
    fill(100, 100, 250, 250, 5);
    apage(2);
    fill(150, 150, 300, 300, 9);
    apage(3);
    fill(200, 200, 350, 350, 15);
    vpage(15);
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 3000; j++)
            ;
        apage(i);
        wipe();
    }
    b_exit(0);
}
```

write

レベル 2

書 式 #include <io.h>

```
size_t write(handle, buffer, count) ;  
int handle; /* 出力するファイルのファイルハンドル */  
void *buffer; /* 出力データの格納領域へのポインタ */  
size_t count; /* 出力バイト数 */
```

機 能

handleで指定されるファイルへbufferからcountバイトを書き込みます。
書き込み処理は、ファイルポインタの現在の位置から始められます。
ファイルを追加用にオープンした場合は、書き込み処理はエンドオブファイル (EOF) から始められます。
書き込み処理の終了後ファイルポインタは、実際に書かれたバイト数だけ進められます。

参 照 関 数

fwrite, open, read

戻 り 値

実際に書き込んだバイト数を返します。
戻り値がcountより小さい値になる場合もあります (たとえばcountバイトを書く前にディスクに空き領域がなくなった場合)。
エラーが発生した場合は-1を返し、errnoに次の値がセットされます。

errno の値	内 容
EACCES	ファイルが読み込み専用
EBADF	ファイルハンドルが無効
ENOSPC	デバイスに空き領域がない

指定されたファイルをテキストモードでオープンした場合、出力時に改行‘\n’は復帰改行‘\r\n’に置き換えられます。

この置き換えは戻り値には影響を与えません。

テキストモードでオープンされているファイルへ書き込む場合に CTRL-Z コード‘\x1A’は、論理的エンドオブファイル (EOF) として扱われます。

またバッファ内に CTRL-Z コード文字があると、デバイスへ CTRL-Z コードを書き込んだ時点で終了します。

プログラム例

```

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <io.h>
#include <stat.h>

#define SIZE 16

main(int argc, char *argv[])
{
    char buf[SIZE];
    char *file;
    int fn;
    int n;

    file = (argc > 1) ? argv[1] : "file";

    if ((fn = open(file, O_RDONLY|O_TEXT)) == -1) {
        printf("can't open %s\n", file);
        exit(1);
    }
    while (!eof(fn)) {
        n = read(fn, buf, SIZE);
        (void)write(fileno(stdout), buf, n);
    }
    close(fn);
    exit(0);
}

```

zentohan

レベル 2

書 式 #include <jfctype.h>

```
int zentohan(c) ;
int c; /* 変換する文字（全角文字） */
```

機 能

全角文字 c を半角文字に変換します。

全角文字は、0x20~0x7E、0xA1~0xDF の半角文字に対応する文字でなければなりません。

全角カタカナも全角ひらがなも、半角カタカナに変換されます。

濁点、半濁点の文字は、8~15 ビット目に ' '、または ' ' のコード、0~7 ビット目に濁点、半濁点なしのカナコードが入ります。

戻 り 値

半角文字を返します。

変換できなかった場合には c の値をそのまま返します。

参 照 関 数

hantozen

プログラム例

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

main()
{
    int ch;
    int c, cc;

    while ((c = getchar()) != EOF) {
        if (iskanji(c)) {
            cc = c;
            ch = (c << 8);
            ch += (c = getchar());
            if (jiskata(ch) || jishira(ch)) {
                putchar(zentohan(ch));
                putchar(zentohan(ch) >> 8);
            }
            else {
                putchar(cc);
                putchar(c);
            }
        }
        else
            putchar(c);
    }
}
```

付 録

ユーザーライブラリの 作り方

ソースファイルの作成

コンパイル

ライブラリファイル

ライブラリの外部宣言

リンク

第 4 回

ユーザーライブラリ の作成

C compiler PRO-68Kでは標準ライブラリ、IOCSコールライブラリ、DOS
コールライブラリ等を提供していますが、これとは別にユーザー自身がオリ
ジナルにライブラリを作成することができます。――
本編では、ユーザーライブラリの作りかたを各ステップごとに述べるととも
に、作成にあたっての注意事項についても述べていきます。

1

ソースファイルの作成

ライブラリもメインプログラムと同様に、エディタでソースファイルを作成します。

エディタの起動方法、および操作手順等については、「Human68k ユーザーズマニュアル」を参照してください。

ソースファイルの作成が終わったら、コンパイルを行います。コンパイルは、CC コマンドを起動することにより行います。

このとき注意しなければならないことは、CC コマンドを起動すると自動的にリンクまで行ってしまうので、リンク作業を抑止しなければならないということです。

というのは、ライブラリのリンクはメインプログラムとともに行う必要があるのですが、ライブラリのコンパイル段階でリンクを行うわけにはいかないからです。

そのようなわけで、ライブラリはオブジェクトコードだけ作成しておけば済みます (XC 標準のライブラリもオブジェクトファイルとして提供されています)。

リンク処理の抑止は、Fc スイッチにより行います。ライブラリのコンパイルを行う場合の、CC コマンドのコマンドラインの具体例を以下に示します。

```
CC /Fc jstrlen.c
```

なお、CC コマンド、および CC コマンドのスイッチの詳細については、「C ユーザーズマニュアル」の「第2章 コンパイル」を参照してください。

3 ライブラリファイル 4

C 標準ライブラリの登録ファイルは `clib.1` です。ユーザーが自身でライブラリを作成した場合、`clib.1` に登録することもできますし、またユーザー自身のファイルに登録することもできます。また、ライブラリはオブジェクトファイル（拡張子 `o`）の形式でよいのですが、ファイルの管理上、ライブラリファイル（拡張子 `1`）に登録しておけば何かと便利です。

ライブラリファイルへの登録方法については、「アセンブルマニュアル」の「第 8 章 ライブラリアン」を参照してください。

4 ライブラリの外部宣言

新規に作成したライブラリをメインプログラムから呼び出す場合、ライブラリの外部宣言、および該当ライブラリで使用する（既存のインクルードファイルでは未定義の）記号定数や構造体などがある場合には、これらの定義が必要となります。

上記の外部宣言および定義を行う方法には、以下の2種類があります。

①新規にインクルードファイル（ユーザーインクルードファイル）を作成し、ここで外部宣言と定義を行う。

②メインプログラム中で外部宣言と定義を行う。

新規に作成したライブラリが汎用性に富んでおり、多数のプログラムで使用される可能性がある場合や、そのライブラリが使用する構造体や記号定数が数多い場合などでは、①の方法が有効でしょうし、そうでない場合には、②で済むこともあります。

これについては、ライブラリの作成者が適宜、選択してください。

5

リンク

ライブラリは、メインプログラムのリンク時にはじめて結合されます。

リンク時には、リンカのコマンドライン上にライブラリファイルの名前を指定します。

以下にコンパイラ、およびリンカのコマンドラインの具体例を示します。

```
cc jstrsample. c jstrlen. o 
```

```
lk jstrsample.o jstrlen.o clib.l floatfnc.l 
```

上記の例では、ライブラリファイル名は、clib.l となっていますが、ユーザーがライブラリファイルを作成した場合には、代わりにそのファイル名を指定してください。

リマ

5

リマは、エーサーラトマリの作りかたに
 リマは、エーサーラトマリの作りかたに
 リマは、エーサーラトマリの作りかたに

cc-javascript
 the-javascript

上記の例では、エーサーラトマリの作りかたに
 エーサーラトマリの作りかたに
 エーサーラトマリの作りかたに

索引 50 音順

韻音 02

50 音順およびアルファベット順の索引は、すべて「C ライブラリマニュアル VOL.1」に対応しています。

ア

アーカイバ	6
アーカイブファイル	6
アクセス	25
アークコサイン	51
アークサイン	51
アークタンジェント	51
アクティブページ	67
アセンブラ	3
アドレス	15
アニメーション	77
アペンド	45
アボート処理	81
アラーム	76
アレイチェーン	78

イ

イジェクト	75
異常終了	53
イメージ制御	17
色コード	68
色モード	67
インクルード	19
インクルードファイル	4、9、12、15、19
引数	3
引数型	9
引教配列	53、55
引数リスト	53、55

エ

英大文字	33
英小文字	33
英数字	33

英文字	33
エコーバック	50
エディタ	10
エラー	46
エラーインジケータ	5、46
エラーコード	5、17
エラー情報	16
エラー処理	5、18
エラーメッセージ	15
円弧	67
円周率	67
エンドオブファイル	40、47
エンドオブファイルインジケータ	5

オ

大文字	34
オブジェクトコード	6
オブジェクトファイル	6
オペレーティングシステム	53
親プロセス	55
音色データ	70
音色番号	70

カ

外字エリア	73
解像度	67
仮引数	3
書き込みモード	45
拡張子	6、9
拡張バッファ操作関数	29
楽譜データ	70
仮数部	51
仮想割りあて	90
画像データ	68

カーソル	66
カテゴリー	92
型	13
カナ句読点	61
カナ文字	61
可変引数	13、18、92
画面サイズの設定	19
画面処理	19
カラー属性	73
カレントドライブ	10
カレントディレクトリ	10
漢字	22
漢字処理関数	61
関数	3
関数宣言	17
関数定義	3
環境時間変数	59
環境テーブル	55
環境変数	10、92
環境文字列	15

キ

キー	66
記号定数	4、44
疑似乱数	65、92
キー入力	19
虚数部	16

ク

クイックソート	56
空白文字	33
句読点	33
グラフィック画面	68
グラフィック制御	17、21
クリッピング	68

グリニッジ標準時	15、59
グローバル変数	5、15、59

コ

構成文字	58
構造体	4、9
コサイン	51
固定長データ項目	41
子プロセス	53、55
コマンド	91
小文字	34
コモン領域	89
コンソール	50
コンソール直接出力	84、85、86
コンソール直接入力	86
コンソール入出力	20
コンソール入出力関数	50
コントラスト	67

サ

最小値	22
最大値	22
再定義可能キー	84
サイン	52
作業用ファイル	42
サーチ	56
サブスレッド	90
算術ライブラリ用関数	18

シ

時間関数	59
時間情報	59
色調補正	68
識別子	3、33
シーク	13

シグナル	54
シグナル割り込み	53
時刻	65、67
時刻地域名	15
指数関数	51
指数部	51
システムコール	15
システム時間	59
システムバッファ	44
自然対数	52
実行速度	3
実数部	16
指定座標	68
指定文字	62
指定文字セット	57、58
シフトキー	72
主スレッド	90
10 進数	33
16 進数	33、64
終了コード	54
受信データ	74
ジョイスティック	18、71
初期化	65
シンボル	22、23、24

ス

数学ライブラリ	22
数値	35
数値演算	5
数値演算関数	51
数値表現形式	35
スクロール	66
スタック環境	92
ステータス情報	23
ストリーム	5、14、40、43
ストリーム入出力関数	13、40

ストリームバッファ	42、44
ストリームポインタ	43
ストリームルーチン	25
スーパーインポーズ	68
スーパーバイザモード	78
スプライト	18、70
スプライト制御	23
スプライトプレーン	70
スプライトパターンデータ	71
スプライトレジスタ	80
スペース	66
スレッド	91
スレッド情報	90

セ

制御文字	33
整数	35
整数型	22
整数値	51
整数部	52
絶対値	51、92
全角英大文字	62
全角英小文字	62
全角英文字	62
全角カタカナ文字	62
全角空白文字	62、63
全角句読点文字	63
全角数字	62
全角ひらがな文字	63
全角文字	58、61、62、63
先行入力	83
先頭文字	33

ソ

属性	38
----	----

50 音順

ソースファイル	4、6
ソート	56
ソート機能	25
ソフトキーボード	77

タ

タイマー	91
タブスキップ	67
タンジェント	52

チ

地方時	15、60
地方時調整	59
チャンネル	69
直角三角形	51

テ

低水準入出力	17
低水準入出力関数	47
低水準入出力操作	21
ディスク	64
デフォルト値	15
ディレクトリ	37
ディレクトリ制御関数	37
ディレクトリ操作	17
テキスト画面	66
テキストモード	46
データ変換	25
データ変換関数	35
デバイスドライバ	82
デバイスドライバの直接入出力	88
テレビ画面	69
電卓	78
テンポ	70

テンポラリファイル	14
-----------	----

ト

トークン	58
時計	76
ドット座標	73
ドライブパラメータブロック	87
ドライブ名	14
トラック	69
トラックデータ	69
トラックバッファ	69

ナ

夏時間	15
夏時間名	15

ニ

2進数	64
日本	15
日本語対応版文字分類関数	22
日本語ライブラリ	61
日本語ライブラリ用関数	18
入出力リダイレクション	44
入出力ルーチン	40、47

ヌ

ヌルポインタ	13、24
ヌル文字	57

ハ

バイト数	12
バイナリサーチ	56

バイナリモード	47
ハイパボリックコサイン	51
ハイパボリックサイン	52
ハイパボリックタンジェント	52
配列	31
パス名	14、38
パターンデータ	70
8進数	33、64
バックグラウンド	71
バックブランクドテキスト	71
バッファ	13
バッファサイズ	24
バッファ操作関数	29、57
バッファリング	40
ハードウェア	21
パレット	73
パレット色	67
パレットコード	68
半角文字	61
半透明制御	69

ヒ

日付	65
ヒープ領域	32
表示可能文字	33
表示画面	67
標準エラー出力	5、44、48
標準型	16
標準出力	42、44
標準入力	41、44
標準プリンタ出力	44
標準補助入出力	44
標準ライブラリ	25

フ

ファイル	13、16、65
ファイル記述子	47
ファイル情報	38
ファイル処理	19
ファイル操作関数	38
ファイルデバイス	45
ファイルハンドル	38、47
ファイルポインタ	41、45、46、48
ファイル名	10、14
ファンクションキー	67
ファンクションコール	17、20
フォーマット	42
副作用	34
複素数	16、51
符号	35
物理フォーマット	75
浮動小数点演算用定数	17
浮動小数点データ	21
浮動小数点値	35、51
ブート情報	78
フラグ	20、81
フラグ定義	23
フラッシュ	40、45
フリーエリア	66
プリプロセッサ	3
プリンタ	67
プリンタポート	75
ブレイク値	31
プロセス	53
プロセス制御関数	18、23、53
分数部	52

へ

平方根	52
-----	----

50 音順

ベクタ	78
ベリファイフラグ	87
変換処理	64
変換モード	15、41
変数	9

ホ

ポインタ	12
ボックス	67

マ

マウス	18、22
マウスカーソル	69
マウスボタン	69
マクロ	3、13、22
マクロ定義	3

メ

メモリサイズ	3
メモリ操作関数	25
メモリブロック	29
メモリ領域	31
メモリ割りあて関数	31

モ

文字型	22
文字コード	64、82
文字デバイス	39
文字チェック	17
文字の分類	33
文字変換	17
文字変換関数	22
文字列	13、35、57

文字列処理	19
文字列操作関数	57
文字分類関数	61
モジュール番号	89
モード	13
戻り値	5

ユ

ユーザーモード	84
ユーザー領域	45

ヨ

曜日	66
読み込みモード	45

ラ

ライブラリアン	6
ライブラリファイル	6
ライブラリルーチン	3
ラストコピー	81

リ

リダイレクション	48
リミットチェック	18
リワインド	46
リンクアレイチェーン	78

レ

レジスタ	70、72
------	-------

論理エラー19、 92
ロングワードデータ30

ワーキングディレクトリ37

ワードデータ列30

割り込みラスタ77

アルファベット順

50 音順およびアルファベット順の索引は、すべて「C ライブラリマニュアル VOL.1」に対応しています。

A

ADPCM	17、19、76、77
ADPCM 制御	72
ANK コード	79
ANSI 規格	20
AR	6
ASCII コード	33、58
ASCII 文字	33
assert. h	17、19、92
assert マクロ	19
audio. h	17、19、64
AUX	82、83

B

BASIC ライブラリ	64
basic. h	17、19、64
basic0. h	17、19、64
BASTOC	64
BCD 表現	76
BG コントロールレジスタ	80
BG スクロールレジスタ	80
BG テキスト	80
BOOLEAN	13
BUFSIZ	13、24
BUF_SIZ	44
BYTE	12

C

C 言語の識別子	33
C ランタイムライブラリ	5
char 型	12
class. h	12、17、19
clearerr 関数	5

CLIB.1	6
CLK_TCK	13
clock 関数	12、13
CLOCKS_PAR_SEC	13
clock_t	12
complex 構造体	16
conio. h	17、20、50
CRT	73
CRTC	77
ctype. h	17、20、21、33

D

DEFAULT	12
#define	3
#defineMACRO	43
direct. h	17、20、37
DMA 転送	78
DMA 転送制御	72
DOS コール	82
doslib. h	17、20、82
double 型	56
div 関数	12
div_t	12

E

EOF	13、24
eof	48
EOS	13
error. h	17、20
exception 構造体	16
exit 関数	13
EXIT_FAILURE	13
EXIT_SUCCESS	13

F

fcntl. h	17、20
fctype. h	17、21、33
feof	46
ferror 関数	5
fgetpos 関数	12
FILE	13
FILE 構造体	25、43
FILENAME_MAX	14
float 型	56
float. h	17、21
FM 音源	18、22、69、77
FM 音源制御	72
FOPEN_MAX	14
fpos_t	12
fseek 関数	13、46
fstat 関数	16、18
ftell	46
ftime 関数	16、18

G

graph. h	17、21、64
----------	----------

H

H-SYNC	77
--------	----

I

# ifdef	9
# ifndef	9
image. h	17、21、64
# include	10
# include <ctype. h>	34
# include <fctpe. h>	34

# include プロセッサ命令	4
INT	12
int 型	12、92
IOCS コール	72
IOCS ルーチン	72
iocslib. h	17、21、72
io. h	17、21、38、47
IPL	81

J

jfcype. h	18、22、61、62
JIS 漢字コード	79
JIS 第一水準	62
JIS 第二水準	62
jmp_buf	13
jstring. h	18、22、62

L

ldiv 関数	12
ldiv_t	12
LED キー	73
LIB	6
limits. h	18、22
LONG	13
long	52
long 型	13、35、56、92
longimp 関数	13
lseek	48
L_tmpnam	14

M

matherr 関数	5
math. h	16、18、22、51
MFP	77

アルファベット順

MIDI	70
MIDI 対応 FM 音源	18、23
MML	70
mouse. h	18、22、64
music. h	18、22、64
music2. h	18、23、64

N

_NFILE	14、25
NULL	13、24

O

OS	81、90
----	-------

P

path 変数	53、54、55
PCM	64
PCG	80
perror 関数	5
printf 関数	24
process. h	18、23、53
ptrdiff_	12

R

rbrk 関数	32
rewind	46
rnd	65
ROM	72
RS-232C	44、48、74、82、83
RS-232C 回線の入出力制御	82

S

sbrk 関数	32
SEEK_CUR	13
SEEK_END	13
SEEK_SET	13
setbuf 関数	24、44
setjmp 関数	13、18
setjmp. h	13、18、23、92
short 型	12、35、42、56、92
signal 関数	18
signal. h	18、23
sizeof	12
size_t	12
sprite. h	18、23、64
stat 構造体	16
stat. h	16、18、23、38
stdarg. h	13、18、24、92
stdaux	44、48
STDC 識別子	9
stddef. h	12、13、18、24、92
stderr	5、44、48
stdin	44、48
stdio. h	13、14、18、24、40、92
stdlib. h	13、15、18、25、31、35、51、53、56、92
stdout	44、48
stdprn	44、48
stick. h	18、25、64
string. h	18、25、57、92
sys_errlist 配列	15

T

tell	48
time 関数	12、18
timeb 構造体	16、59

timeb. h	16、18、26、59	void 型	12
time. h	15、16、18、26、59		
time_t	12		
time_t 型	59、60		
TIMER-D	77		
tm 構造体	16、59、60		
tmpfile 関数	14		
TMP_MAX	14		
tmpname 関数	14		
toupper ルーチン	3		
TV コントロールコード	73		
TV の表示制御	72		
U			
UBYTE	12		
UINT	12		
ULONG	13		
#undef	10		
#undef MACRO	43		
#undef tolower	34		
unsigned char 型	12		
unsigned int 型	12		
unsigned long 型	13、35		
unsigned short 型	12		
utime 関数	16、18		
utimebuf 構造体	16		
utime. h	16、18、26、59		
UWORD	12		
V			
va_arg マクロ関数	13		
va_list	13		
va_start マクロ関数	13		
V-DISP	77		
VOID	12		
W			
WORD	12		
X			
X 座標	66		
Y			
Y 座標	66		
YM2151	77		

関数索引 アルファベット順

※小文字関数は「C ライブラリマニュアル VOL.1」、大文字関数は「C ライブラリマニュアル VOL.2」のページ数を表記してあります。

A

abort	95
ABORTJOB	3
ABORTRST	4
abs	96
access	97
a_cont	99
acos	100
ADPCMAIN	5
ADPCMAOT	7
ADPCMINP	9
ADPCMLIN	11
ADPCMLOT	13
ADPCMMOD	15
ADPCMOUT	16
ADPCMSNS	18
a_end	101
AKCONV	19
ALARMGET	20
ALARMMOD	21
ALARMSET	22
ALLCLOSE	24
allmem	153
apage	102
APAGE	25
a_play	103
a_rec	106
asc	105
asctime	108
asin	109
assert	110
a_stat	111
a_stop	113
atan	114

atan2	114
atexit	115
atof	116
atoi	116
atol	116
atow	116

B

B_BADFMT	26
b_binS	118
B_BPEEK	27
B_BPOKE	28
b_chrS	119
B_CLR_AL	29
B_CLR_ED	30
B_CLR_ST	31
B_COLOR	32
B_CONSOL	33
b_csw	120
B_CUROFF	34
B_CURON	35
b_dateS	121
b_dayS	122
B_DEL	36
B_DOWN	37
B_DOWN_S	38
B_DRVCHK	39
B_DRVSNS	40
B_DSKINI	41
beep	123
B_EJECT	42
B_ERA_AL	43
B_ERA_ED	44
B_ERA_ST	45

b_exit	124	b_inkeyS	148
b_fclose	125	b_input	149
b_fcloseall	126	B_INS	59
b_feof	127	b_int	150
b_fgetc	128	B_INTVCS	60
b_flprint	129	b_iprint	151
b_fopen	130	b_itoa	152
B_FORMAT	46	BITSNS	61
b_fprint	131	B_KEYINP	62
b_fputc	132	B_KEYSNS	63
b_fread	133	bldmem	153
b_freads	134	B_LEFT	64
b_free	135	b_leftS	154
b_fseek	136	b_linput	155
b_fwrite	137	B_LOCATE	65
b_fwrites	138	B_LPEEK	66
BGCTRLGT	47	B_LPOKE	67
BGCTRLST	48	B_MEMSET	68
bg_fill	139	B_MEMSTR	69
bg_get	140	b_midS	156
bg_put	141	b_mirrorS	157
BGSCRLGT	49	b_octS	158
BGSCRLST	50	BOOTINF	70
bg_scroll	142	box	159
bg_set	143	BOX	72
bg_stat	144	b_pi	161
BGTEXTCL	51	B_PRINT	73
BGTEXTGT	52	B_PUTC	74
BGTEXTST	53	B_PUTMES	75
b_hexS	145	B_READ	76
b_ilprint	129	B_READDI	77
BINDATEBCD	54	B_READDL	78
BINDATEGET	56	BREAKCK	79
BINDATESET	57	B_RECALI	80
BINDNO	58	B_RIGHT	81
b_init	146	b_rightS	162
b_inkey0	147	bsearch	163

アルファベット順

B_SEEK	82	C_COLOR	97
b_setdate	165	C_CUROFF	98
b_settime	166	C_CURON	99
B_SFTSNS	83	C_DEL	100
b_slprint	129	C_DOWN	101
b_spaceS	167	C_DOWN_S	102
b_sprint	168	ceil	181
b_stradd	169	C_ERA_AL	103
b_strchr	170	C_ERA_ED	104
b_strcmp	171	C_ERA_ST	105
b_strfS	173	C_FNKMOD	106
b_stringS	175	cgets	182
b_striS	174	CHANGE_PR	107
b_strncpy	171	chdir	183
b_strrchr	170	CHDIR	108
b_strtok	170	CHGDRV	109
B_SUPER	84	chgfa	184
b_timeS	176	chgft	185
b_tlprint	129	child	187
b_tpalet	177	chkml	188
b_tprint	178	chmod	189
B_UP	85	CHMOD	110
B_UP_S	86	chsize	190
BUS_ERR	87	C_INS	111
B_VERIFY	89	CINSNS	112
B_WPEEK	90	circle	191
B_WPOKE	91	CIRCLE	113
B_WRITE	92	clearerr	193
B_WRITED	93	C_LEFT	114
		CLIPPUT	115
		C_LOCATE	116
		clock	194
		close	195
		CLOSE	117
		clrerr	193
		cls	196
		color	197
C			
cabs	179		
calloc	180		
C_CLS_AL	94		
C_CLS_ED	95		
C_CLS_ST	96		

COMINP118
 COMMON_CK119
 COMMON_DEL120
 COMMON_FRE121
 COMMON_LK123
 COMMON_RD125
 COMMON_WT127
 COMOUT129
 CONSNS130
 console198
 contrast199
 CONTRAST131
 cos200
 cosh200
 COUTSNS132
 C_PRINT133
 cprintf201
 C_PUTC134
 cputs202
 creat203
 CREATE135
 C_RIGHT136
 crt204
 CRTCRAS137
 CRTMOD138
 cscanf205
 csrlin206
 ctime207
 C_UP139
 C_UP_S140
 CURDIR141
 CURDRV142
 C_WIDTH143
 C_WINDOW144

D

DAKJOB145
 DATEASC146
 DATEBIN148
 DATECNV150
 DAYASC151
 DEFCHR152
 DELETE153
 DENSNS154
 difftime208
 DISKRED155
 DISKRED2157
 DISKWRT159
 DISKWRT2161
 div209
 DMAMODE163
 DMAMOV_A164
 DMAMOVE166
 DMAMOV_L168
 dqsort210
 DRVCTRL170
 DRVXCHG172
 dskf211
 DSKFRE173
 dtasc212
 dtcnv214
 dup215
 DUP174
 DUP0175
 dup2215
 DUP2176

E

ecvt217
 eof219

アルファベット順

except	224
EXEC2	177
execl	221
execle	221
execlp	221
execlpe	221
EXECONLY	179
execv	221
execve	221
execvp	221
execvpe	221
exit	225
_exit	225
EXIT	180
EXIT2	181
exp	226

F

fabs	227
FATCHK	182
FATCHK2	183
fclose	228
fcloseall	228
fcvt	230
fdelete	232
fdopen	233
feof	234
ferror	235
fflush	236
FFLUSH	185
fgetc	238
FGETC	186
fgetchar	238
fgetpos	239
fgets	240
FGETS	187

FILEDATE	188
filelength	241
fileno	242
files	243
FILES	190
fill	246
FILL	192
fix	247
floor	248
flushall	249
fmod	250
fmode	251
FNCKEYGT	193
FNCKEYST	195
FNTGET	197
fopen	252
fprintf	254
fputc	255
FPUTC	199
fputchar	255
fputs	256
FPUTS	200
fqsort	210
fread	257
free	259
frename	260
freopen	261
frexp	262
fscanf	263
fseek	265
fsetpos	267
fstat	268
ftell	270
ftime	271
ftpack	272
ftunpk	274
fwrite	276

G

G_CLR_ON	201
gcvt	277
get	278
GETASSIGN	202
getc	280
GETC	203
getch	282
getchar	280
GETCHAR	204
getche	283
getcwd	284
GETDATE	205
GETDPB	206
getenv	286
GETENV	208
getfa	287
getft	289
GETGRM	209
getl	290
getmem	291
getml	291
GETPDB	210
getpid	293
GET_PR	212
gets	294
GETS	214
GETSS	215
GETTIM2	216
GETTIME	217
getw	295
gmtime	296
GPALET	218

H

HANJOB	219
hantozen	298
HENDSPIC	220
HENDSPIO	221
HENDSPIP	222
HENDSPIR	223
HENDSPMC	224
HENDSPMO	225
HENDSPMP	226
HENDSPMR	227
HENDSPSC	228
HENDSPSO	229
HENDSPSP	230
HENDSPSR	231
home	299
HOME	232
hsv	300
HSVTORGB	233
HSYNCST	234
hypot	302

I

img_color	303
img_home	304
img_ht	305
img_load	306
img_pos	307
img_put	308
img_save	309
img_scrn	310
img_set	311
img_still	312
INDOSFLG	235
INIT_PRN	236

アルファベット順

INKEY	237
INP232C	238
INPOUT	239
instr	313
INTVCG	240
INTVCS	241
IOCTRLDVCTL	242
IOCTRLDVGT	243
IOCTRLFDCTL	244
IOCTRLFDGT	245
IOCTRLGT	246
IOCTRLIS	248
IOCTRLOS	249
IOCTRLRD	250
IOCTRLRH	251
IOCTRLRTSET	252
IOCTRLST	253
IOCTRLWD	254
IOCTRLWH	255
IPLERR	256
isalkana	314
isalnmkana	314
isalnum	315
isalpha	315
isascii	315
isatty	317
isctrl	315
iscsym	315
iscsymf	315
isdigit	315
isgraph	315
isgrkana	314
iskana	318
iskanji	319
iskanji2	319
iskmoji	318
iskpun	318

islower	315
ISNS232C	257
isodigit	315
ispnkana	314
isprint	315
isprkana	314
ispunct	315
isspace	315
isupper	315
isxdigit	315
itoa	320

J

jisalpha	321
jisdigit	321
jishira	321
jiskata	321
jiskigou	321
jisl0	322
jisl1	322
jisl2	322
jislower	321
JISSFT	258
jisspace	321
jisupper	321
jiszen	322
JOYGET	259
jstrchr	323
jstrcmp	324
jstrncmp	325
jstrrchr	326

K

kbhit	327
KEEPPR	260

key	328
keysns	329
KEYSNS	261
KFLUSHGC	262
KFLUSHGP	263
KFLUSHGS	264
KFLUSHIN	265
KFLUSHIO	266
KILL_PR	267
K_INSMOD	268
K_KEYBIT	269
K_KEYINP	271
K_KEYSNS	272
K_SFTSNS	273

L

labs	96
ldexp	330
ldiv	331
LEDMOD	274
line	332
LINE	275
LOAD	276
LOADEXEC	277
LOADONLY	278
localtime	334
locate	335
LOCK	279
LOF232C	280
log	336
log10	336
longjmp	337
lqsort	210
lseek	338
ltoa	340

M

MAKEASSIGN	281
MAKETMP	282
malloc	342
m_alloc	343
MALLOC	284
MALLOC2	285
m_assign	345
matherr	346
max	348
m_cont	349
md_cont	350
md_init	351
md_off	352
md_on	353
md_play	354
md_regr	355
md_regw	356
md_stat	357
md_stop	358
md_wrt	359
memccpy	360
memchr	361
memcmp	362
memcmpi	363
memcpy	365
memicmp	363
memmove	365
memset	367
m_free	368
MFREE	286
min	369
m_init	370
mkdir	371
MKDIR	287
mktemp	372

アルファベット順

mktime	375
modf	376
mouse	377
MOVE	288
movedata	365
movmem	365
m_play	378
msarea	380
msbtn	382
MS_CURGT	290
MS_CUROF	291
MS_CURON	292
MS_CURST	293
MS_GETDT	294
MS_INIT	295
MS_LIMIT	296
MS_OFFTM	297
MS_ONTM	298
MS_PATST	299
mspos	384
MS_SEL	301
MS_SEL2	302
MS_STAT	304
msstat	385
m_stat	387
m_stop	388
m_tempo	389
m_trk	390
m_vget	391
m_vset	392

N

NAMECK	305
NAMESTS	306
NEWFILE	308
nfiles	393

NFILES	309
--------	-----

O

offsetof	395
ONTIME	310
open	396
OPEN	311
OPEN_PR	312
OPMINTST	314
OPMSET	315
OPMSNS	316
OS_CUROF	317
OS_CURON	318
OSNS232C	319
OUT232C	320
OUTLPT	321
OUTPRN	322

P

paint	399
PAINT	323
palet	400
PATHCHK	324
perror	401
pi	402
point	403
POINT	326
pos	404
pow	405
PRINT	327
printf	406
PRNINTST	328
PRNOUT	329
PRNSNS	330
pset	411

PSET331
 PSPSET332
 put412
 putc414
 putchar416
 putchar414
 PUTCHAR334
 putenv417
 PUTGRM335
 putl419
 puts420
 putw421

Q

qsort422

R

raise424
 rand426
 randomize427
 RASSIGN336
 rbrk428
 read429
 READ337
 realloc431
 remove547
 rename433
 RENAME338
 repmem435
 rewind436
 rgb437
 rlsmem291
 rlsm1291
 RMACNV339
 rmdir438

RMDIR340
 rmtmp439
 rmtmp439
 rnd440
 ROMVER341
 rstmem441

S

sbrk442
 scanf443
 screen447
 SCROLL342
 SEEK343
 SEND_PR344
 SET232C346
 SETBLOCK348
 setbuf449
 SETDATE349
 SETENV350
 setjmp451
 setmem367
 setmode453
 setmspos455
 setnbf456
 SETPDB351
 SETTIM2352
 SETTIME353
 setvbuf456
 SFTJIS354
 sgn457
 signal458
 sin461
 sinh461
 sizmem462
 SKEY_MOD355
 SKEYSET357

アルファベット順

SLEEP_PR	358	stcgfe	484
S_MALLOC	359	stcgfn	484
S_MFREE	360	stick	485
SNSPRN	361	strbpl	486
SPALET	362	strcat	487
spawnl	463	strchr	487
spawnle	463	strcmp	487
spawnlp	463	strcmpi	487
spawnv	463	strcpy	487
spawnve	463	strcsn	487
spawnvp	463	strdup	487
SP_CGCLR	363	strerror	491
sp_clr	465	strftime	492
sp_color	466	stricmp	487
sp_def	467	strig	495
SP_DEFCG	364	strins	496
sp_disp	468	strlen	497
SP_GTPCG	365	strlwr	498
SP_INIT	366	strmfe	499
sp_init	469	strmfn	500
sp_move	470	strmfp	500
SP_OFF	367	strncat	501
sp_off	472	strncmp	501
SP_ON	368	strncpy	501
sp_on	473	strnset	501
sp_pat	474	strpbrk	504
SP_REGGT	369	strrchr	505
SP_REGST	370	strrev	506
sprintf	476	strset	507
S_PROCESS	371	strsfn	508
sp_set	477	strspn	509
sp_stat	478	strsrt	510
sqrt	479	strstr	511
sqsort	210	strtod	512
srand	480	strtok	514
sscanf	481	strtol	515
stat	482	strtoul	517

strupr519
 SUPER372
 SUPER_JSR373
 SUSPEND_PR374
 swab520
 saww522
 swmem523
 symbol524
 SYMBOL375
 system526

T

tan527
 tanh527
 TCOLOR376
 tell528
 tempfile529
 tempnam530
 TEXTGET377
 TEXTPUT378
 TGUSEMD379
 time532
 TIMEASC381
 TIMEBCD382
 TIMEBIN384
 TIMECNV385
 TIMEGET386
 TIME_PR387
 TIMERDST388
 TIMESSET389
 tmasc533
 tmcnv534
 tmpfile535
 tmpnam536
 toascii538
 _tolower538

_tolower538
 _toupper538
 toupper538
 TPALET390
 TPALET2391
 tqsort210
 TRAP15392
 TVCTRL394
 TXBOX396
 TXFILL397
 TXRASCOPY398
 TXREV399
 TXXLINE400
 TXYLINE401
 tzset540

U

uitoa542
 ultoa543
 ungetc544
 ungetch546
 unlink547
 UNLOCK402
 using548
 utime549
 uwtoa550

V

va_arg551
 va_end551
 val553
 va_start551
 vprintf555
 v_cut554
 VDISPST403

アルファベット順

VERIFY	404
VERIFYG	405
VERNUM	406
vfprintf	555
vpage	557
VPAGE	407
vprintf	555
vsprintf	555

W

wabs	96
WAIT	408
wait	558
width	559
window	560
WINDOW	409
wipe	561
WIPE	410
write	562
WRITE	411
wtoa	550

Z

zentohan	564
----------------	-----

V

vsprintf	555
vprintf	555
vpage	557
VPAGE	407
VERNUM	406
VERIFYG	405
VERIFY	404

T

tolower	538
toascii	538
tmpnam	536
tmpfile	535
tmpcnv	534
tmpasc	533
timeset	389
timerdst	388
time_pr	387
timeget	386
timecnv	385
timebin	384
timebcd	383
timeasc	381
time	333
tcusemd	379
textput	378
textget	377
tempnam	530
tempfile	529
temp	528
tcset	376
tcget	375
tcset	374
tcget	373
tcset	372
tcget	371
tcset	370
tcget	369
tcset	368
tcget	367
tcset	366
tcget	365
tcset	364
tcget	363
tcset	362
tcget	361
tcset	360
tcget	359
tcset	358
tcget	357
tcset	356
tcget	355
tcset	354
tcget	353
tcset	352
tcget	351
tcset	350
tcget	349
tcset	348
tcget	347
tcset	346
tcget	345
tcset	344
tcget	343
tcset	342
tcget	341
tcset	340
tcget	339
tcset	338
tcget	337
tcset	336
tcget	335
tcset	334
tcget	333
tcset	332
tcget	331
tcset	330
tcget	329
tcset	328
tcget	327
tcset	326
tcget	325
tcset	324
tcget	323
tcset	322
tcget	321
tcset	320
tcget	319
tcset	318
tcget	317
tcset	316
tcget	315
tcset	314
tcget	313
tcset	312
tcget	311
tcset	310
tcget	309
tcset	308
tcget	307
tcset	306
tcget	305
tcset	304
tcget	303
tcset	302
tcget	301
tcset	300
tcget	299
tcset	298
tcget	297
tcset	296
tcget	295
tcset	294
tcget	293
tcset	292
tcget	291
tcset	290
tcget	289
tcset	288
tcget	287
tcset	286
tcget	285
tcset	284
tcget	283
tcset	282
tcget	281
tcset	280
tcget	279
tcset	278
tcget	277
tcset	276
tcget	275
tcset	274
tcget	273
tcset	272
tcget	271
tcset	270
tcget	269
tcset	268
tcget	267
tcset	266
tcget	265
tcset	264
tcget	263
tcset	262
tcget	261
tcset	260
tcget	259
tcset	258
tcget	257
tcset	256
tcget	255
tcset	254
tcget	253
tcset	252
tcget	251
tcset	250
tcget	249
tcset	248
tcget	247
tcset	246
tcget	245
tcset	244
tcget	243
tcset	242
tcget	241
tcset	240
tcget	239
tcset	238
tcget	237
tcset	236
tcget	235
tcset	234
tcget	233
tcset	232
tcget	231
tcset	230
tcget	229
tcset	228
tcget	227
tcset	226
tcget	225
tcset	224
tcget	223
tcset	222
tcget	221
tcset	220
tcget	219
tcset	218
tcget	217
tcset	216
tcget	215
tcset	214
tcget	213
tcset	212
tcget	211
tcset	210
tcget	209
tcset	208
tcget	207
tcset	206
tcget	205
tcset	204
tcget	203
tcset	202
tcget	201
tcset	200
tcget	199
tcset	198
tcget	197
tcset	196
tcget	195
tcset	194
tcget	193
tcset	192
tcget	191
tcset	190
tcget	189
tcset	188
tcget	187
tcset	186
tcget	185
tcset	184
tcget	183
tcset	182
tcget	181
tcset	180
tcget	179
tcset	178
tcget	177
tcset	176
tcget	175
tcset	174
tcget	173
tcset	172
tcget	171
tcset	170
tcget	169
tcset	168
tcget	167
tcset	166
tcget	165
tcset	164
tcget	163
tcset	162
tcget	161
tcset	160
tcget	159
tcset	158
tcget	157
tcset	156
tcget	155
tcset	154
tcget	153
tcset	152
tcget	151
tcset	150
tcget	149
tcset	148
tcget	147
tcset	146
tcget	145
tcset	144
tcget	143
tcset	142
tcget	141
tcset	140
tcget	139
tcset	138
tcget	137
tcset	136
tcget	135
tcset	134
tcget	133
tcset	132
tcget	131
tcset	130
tcget	129
tcset	128
tcget	127
tcset	126
tcget	125
tcset	124
tcget	123
tcset	122
tcget	121
tcset	120
tcget	119
tcset	118
tcget	117
tcset	116
tcget	115
tcset	114
tcget	113
tcset	112
tcget	111
tcset	110
tcget	109
tcset	108
tcget	107
tcset	106
tcget	105
tcset	104
tcget	103
tcset	102
tcget	101
tcset	100
tcget	99
tcset	98
tcget	97
tcset	96
tcget	95
tcset	94
tcget	93
tcset	92
tcget	91
tcset	90
tcget	89
tcset	88
tcget	87
tcset	86
tcget	85
tcset	84
tcget	83
tcset	82
tcget	81
tcset	80
tcget	79
tcset	78
tcget	77
tcset	76
tcget	75
tcset	74
tcget	73
tcset	72
tcget	71
tcset	70
tcget	69
tcset	68
tcget	67
tcset	66
tcget	65
tcset	64
tcget	63
tcset	62
tcget	61
tcset	60
tcget	59
tcset	58
tcget	57
tcset	56
tcget	55
tcset	54
tcget	53
tcset	52
tcget	51
tcset	50
tcget	49
tcset	48
tcget	47
tcset	46
tcget	45
tcset	44
tcget	43
tcset	42
tcget	41
tcset	40
tcget	39
tcset	38
tcget	37
tcset	36
tcget	35
tcset	34
tcget	33
tcset	32
tcget	31
tcset	30
tcget	29
tcset	28
tcget	27
tcset	26
tcget	25
tcset	24
tcget	23
tcset	22
tcget	21
tcset	20
tcget	19
tcset	18
tcget	17
tcset	16
tcget	15
tcset	14
tcget	13
tcset	12
tcget	11
tcset	10
tcget	9
tcset	8
tcget	7
tcset	6
tcget	5
tcset	4
tcget	3
tcset	2
tcget	1

機能別関数索引

※小文字関数は「C ライブラリマニュアル VOL.1」、大文字関数は「C ライブラリマニュアル VOL.2」のページ数を表記してあります。

バッファ操作

関数名	機能	ページ
memccpy	バッファ内文字列を他のバッファにコピーする	360
memchr	バッファ内の文字列を検索する	361
memcmp	2つのバッファ内の文字列を比較する	362
memcmpi	2つの領域を英大/小文字を区別せずに比較する	363
memcpy	バッファ内文字列を他のバッファにコピーする	365
memicmp	2つの領域を英大/小文字を区別せずに比較する	363
memmove	指定領域の内容を他へ転送する	365
memset	バッファ内を指定文字列で初期化する	367
movedata	バッファ内文字列を他のバッファにコピーする	365
movmem	バッファ内文字列を他のバッファにコピーする	365
repmem	指定メモリブロックを複数回コピーする	435
setmem	バッファ内を指定文字列で初期化する	367
swmem	2つのメモリブロックを入れ替える	523
swab	データの上位/下位バイトを入れ替える	520
swaw	データの上位/下位ワードを入れ替える	522

メモリ割りあて

関数名	機能	ページ
allmem	使用可能なすべてのメモリを確保する	153
bldmem	メモリブロックをキロバイト単位で確保する	153
calloc	配列の領域を割りあてる	180
chkml	未使用メモリブロックの最大のサイズ (バイト) を求める	188
free	割りあて済みのメモリブロックを解放する	259
getmem	指定サイズのメモリを確保する	291
getml	指定サイズのメモリを確保する	291
malloc	メモリブロックを割りあてる	342
rbrk	ブレイク値を初期状態に戻す	428
realloc	割りあて済みのメモリブロックのサイズを変更する	431
rlsmem	確保したメモリを解放する	291
rlsml	確保したメモリを解放する	291
rstmem	使用しているメモリブロックをすべて解放する	441

sbrk	ブレイク値をリセットしてヒープ領域を拡張する	442
sizmem	使用可能なメモリブロックをサイズ（ワード）を求める	462

文字の分類と変換

関数名	機 能	ページ
isalnum	英数字の判定を行う	315
isalpha	英文字の判定を行う	315
isascii	ASCII 文字の判定を行う	315
iscsym	識別子の判定を行う	315
iscsymf	識別子の先頭文字の判定を行う	315
isctrl	制御文字の判定を行う	315
isdigit	10 進数の判定を行う	315
isgraph	表示可能文字の判定を行う	315
islower	英小文字の判定を行う	315
isodigit	8 進数の判定を行う	315
isprint	表示可能文字の判定を行う	315
ispunct	句読点の判定を行う	315
isspace	空白文字の判定を行う	315
isupper	英大文字の判定を行う	315
isxdigit	16 進数の判定を行う	315
toascii	文字を ASCII コードへ変換する	538
tolower	英大文字を英小文字へ変換する	538
toupper	英小文字を英大文字へ変換する	538
_tolower	英大文字を英小文字へ変換する	538
_toupper	英小文字を英大文字へ変換する	538

データ変換

関数名	機 能	ページ
atof	数値文字列を倍精度浮動小数点値に変換する	116
atoi	数値文字列を整数値に変換する	116
atol	数値文字列を long 型整数値に変換する	116
atow	数値文字列を short 型整数値に変換する	116
ecvt	倍精度浮動少数点値を数値文字列に変換する	217
fcvt	倍精度浮動小数点値を数値文字列に変換する	230
gcvt	倍精度浮動小数点値を数値文字列に変換する	277

itoa	整数を文字列に変換する	320
ltoa	long 型整数を文字列に変換する	340
strtod	文字列を倍精度の浮動小数点値に変換する	512
strtoul	X 進数の表す文字列を符号なし整数に変換する	517
strtol	数値文字列を long 型整数に変換する	515
uiota	符号なし整数を文字列に変換する	542
ultoa	符号なし long 型整数を文字列に変換する	543
uwttoa	符号なし short 型整数を文字列に変換する	550
wtoa	short 型整数文字列に変換する	550

ディレクトリ操作

関数名	機能	ページ
chdir	ワーキングディレクトリを変更する	183
getcwd	ワーキングディレクトリを求める	284
mkdir	サブディレクトリを作成する	371
rmdir	サブディレクトリを削除する	438

ファイル操作

関数名	機能	ページ
access	ファイルの許可設定を調べる	97
chgfa	ファイルの属性を変更する	184
chgft	ファイルの作成日時を変更する	185
chmod	ファイルの許可設定を変更する	189
chsize	ファイルの大きさを変更する	190
filelength	ファイルの長さを調べる	241
files	ファイルを検索しファイルの情報を求める	243
fstat	ファイルハンドルのステータス情報を求める	268
getfa	ファイルの属性を調べる	287
getft	ファイルの作成日時を調べる	289
isatty	文字デバイスかどうか調べる	317
mktemp	仮のファイル名を作成する	372
nfiles	次のファイルの候補を検索しファイルの情報を求める	393
rename	ファイル名を変更する	433
remove	ファイルを削除する	547
setmode	ファイルの変換モードを設定する	453

stat	ファイル名からファイルのステータス情報を求める	482
strsn	ファイル名をドライブ名やパス名などに分解する	508
unlink	ファイルを削除する	547
dtasc	日付情報を文字列に変換する	212
dtcnv	日付を表す文字列を日付情報に変換する	214
ftpack	年月日時分秒の情報を日時情報に変換する	272
ftunpk	日時情報を年月日時分秒の情報に変換する	274
tmsc	時刻情報を文字列に変換する	533
tmcnv	時刻を表す文字列を時刻情報に変換する	534
utime	ファイル更新日付を設定する	549

ストリーム入出力

関数名	機能	ページ
clearerr	ストリームのエラーインジケータをクリアする	193
clrerr	ストリームのエラーインジケータをクリアする	193
fclose	ストリームをクローズする	228
fcloseall	すべてのストリームをクローズする	228
fdopen	ファイルハンドルでストリームをオープンする	233
feof	指定ストリームが EOF かどうか調べる	234
ferror	ストリームの入出力時のエラーを調べる	235
fflush	ストリームをフラッシュする	236
fgetc	ストリームから 1 文字読み込む	238
fgetchar	標準入力から 1 文字読み込む	238
fgetpos	ストリームのアクセス位罫を求める	239
fgets	ストリームから文字列を読み込む	240
fileno	ストリームのファイルハンドルを求める	242
flushall	すべてのストリームをフラッシュする	249
fmode	ファイルの変換モードを変更する	251
fopen	ストリームをオープンする	252
fprintf	ストリームへフォーマットデータを書き込む	254
fputc	ストリームへ 1 文字書き込む	255
fputchar	標準出力へ 1 文字書き込む	255
fputs	ストリームへ文字列を書き込む	255
fread	ストリームから固定長データを読み込む	257
freopen	ストリームポインタを再割りつけする	261
fscanf	ストリームからフォーマットデータを読み込む	263

fseek	ストリームポインタを移動する	265
fsetpos	ストリームのアクセス位置を変更する	267
ftell	ファイルポインタの位置を求める	270
fwrite	ストリームへ固定長データを書き込む	276
getc	ストリームから1文字読み込む	280
getchar	標準入力から1文字読み込む	280
getl	ストリームから int 型データを読み込む	290
gets	標準入力から1行読み込む	294
getw	ストリームから short 型データを読み込む	295
printf	標準出力へフォーマットデータを書き込む	406
putc	ストリームへ1文字書き込む	414
putchar	標準出力へ1文字書き込む	414
putl	ストリームへ int 型データを書き込む	419
puts	標準出力へ1行書き込む	420
putw	ストリームへ short 型データを書き込む	421
rewind	ファイルポインタをファイルの先頭へ移動する	436
scanf	標準出力からフォーマットデータを読み込む	443
setbuf	ストリームバッファの制御を行う	449
setnbf	ストリームバッファを変更する	456
setvbuf	ストリームバッファを変更する	456
sprintf	文字列へフォーマットデータを書き込む	476
sscanf	文字列からフォーマットデータを読み込む	481
ungetc	読み込んだ文字を入力ストリームに戻す	544
vfprintf	ポインタで指定した引数をストリームへ出力する	555
vprintf	ポインタで指定した引数を標準出力へ出力する	555
vsprintf	ポインタで指定した引数を文字列へ出力する	555
rmtemp	テンポラリファイルを削除する	439
rmtmp	テンポラリファイルを削除する	439
tempnam	テンポラリファイルのファイル名を作成する	530
tmpnam	既存ファイル名とは異なるファイル名を作成する	536
tempfile	テンポラリファイルをバイナリモードでオープンする	529
tmpfile	テンポラリファイルをバイナリモードでオープンする	535

低水準入出力

関数名	機能	ページ
close	ファイルをクローズする	195

creat	ファイルを新規に作成する	203
dup	ファイルハンドルをコピーする	215
dup2	ファイルハンドルを強制的に再割りつける	215
eof	ファイルが EOF かどうか調べる	219
lseek	ファイルポインタを移動する	338
open	ファイルをオープンする	396
read	ファイルからデータを読み込む	429
tell	ファイルポインタの位置を求める	528
write	ファイルヘデータを書き込む	562

コンソール入出力

関数名	機能	ページ
cgets	コンソールから文字列を読み込む	182
cprintf	コンソールへフォーマットデータを書き出す	201
cputs	コンソールへ文字列を書き出す	202
cscanf	コンソールからフォーマットデータを読み込む	205
getch	コンソールから 1 文字読み込む	282
getche	コンソールから 1 文字読み込みエコーバックする	283
kbhit	キー入力の有無を調べる	327
putch	コンソールへ 1 文字書き出す	416
ungetch	読み込んだ文字をコンソールに戻す	546
vcprintf	ポインタで指定した引数をコンソールへ出力する	555

数値演算

関数名	機能	ページ
acos	アークコサインを求める	100
asin	アークサインを求める	109
atan	アークタンジェントを求める	114
atan2	アークタンジェントを求める	114
cabs	複素数の絶対値を求める	179
ceil	最小の整数値を求める	181
cos	コサインを求める	200
cosh	ハイパボリックコサインを求める	200
div	int 型符号つき整数の除算を商と余りを求める	209
except	例外処理を行う	224

exp	指数関数を求める	226
fabs	浮動小数点値の絶対値を求める	227
floor	最大の整数値を求める	248
fmod	剰余を求める	250
frexp	浮動小数点値を仮数部と指数部に分ける	262
hypot	直角三角形の斜辺の長さを求める	302
ldexp	浮動小数点値の指数部を求める	330
ldiv	long 型符号つき整数の除算を行い商と余りを求める	331
log	自然対数を求める	336
log10	10 を底とした対数を求める	336
matherr	エラー処理を行う	346
modf	浮動小数点値を整数部と小数部に分ける	376
pow	べき乗を求める	405
sin	サインを求める	461
sinh	ハイパボリックサインを求める	461
sqrt	平方根を求める	479
tan	タンジェントを求める	527
tanh	ハイパボリックタンジェントを求める	527

プロセス制御

関数名	機能	ページ
abort	プロセスを異常終了する	95
atexit	プログラム終了時に呼び出す関数を登録する	115
execl	引数リストによって子プロセスを実行する	221
execle	引数リストと環境によって子プロセスを実行する	221
execlp	引数リストと path 変数によって子プロセスを実行する	221
execlepe	引数リスト、path 変数、環境によって子プロセスを実行する	221
execv	引数配列によって子プロセスを実行する	221
execve	引数配列と環境によって子プロセスを実行する	221
execvp	引数配列と path 変数によって子プロセスを実行する	221
execvpe	引数配列、path 変数、環境によって子プロセスを実行する	221
exit	プロセスを終了する	225
_exit	バッファをフラッシュせずに子プロセスを終了する	225
getpid	プロセス ID を求める	293
raise	プログラムに指定したシグナルを送る	424
signal	シグナル割り込み操作を行う	458

spawnl	引数リストによって子プロセスを実行する	463
spawnle	引数リストと環境によって子プロセスを実行する	463
spawnlp	引数リストと path 変数によって子プロセスを実行する	463
spawnv	引数配列によって子プロセスを実行する	463
spawnve	引数配列と環境によって子プロセスを実行する	463
spawnvp	引数配列と PATH 変数によって子プロセスを実行する	463
system	Human68k のコマンドを実行する	526
wait	直前に実行した子プロセスの終了コードを調べる	558

サーチとソート

関数名	機能	ページ
bsearch	バイナリサーチを実行する	163
dqsort	double 型のデータをソートする	210
fqsort	float 型のデータをソートする	210
lqsort	long 型のデータをソートする	210
qsort	クイックソートを実行する	422
sqsort	short 型のデータをソートする	210
tqsort	ポインタ型のデータをソートする	210

文字列操作

関数名	機能	ページ
strcat	文字列を追加する	487
strchr	指定文字の最初の位置を調べる	487
strcmp	2つの文字列を比較する	487
strcmpi	2つの文字列を比較する	487
strcpy	文字列をコピーする	487
strcspn	指定文字列の最初の位置を調べる	487
strdup	文字列をコピーする	487
stricmp	2つの文字列を英大/小文字を区別せずに比較する	487
strlen	文字列の長さを調べる	497
strlwr	すべての英大文字を英小文字に変換する	498
strncat	文字列を追加する	501
strncmp	2つの文字列を比較する	501
strncpy	文字列をコピーする	501
strnset	指定文字列に置き換える	501

strpbrk	指定文字列を探す	504
strchr	指定文字の最後の位置を調べる	505
strrev	文字列を逆順にする	506
strset	指定文字列に置き換える	507
strspn	指定文字列以外の最初の位置を調べる	509
strstr	char 型引数 1 から char 型引数 2 を検索する	511
strtok	トークンを探す	514
strupr	すべての英小文字を英大文字に変換する	519
strins	文字列を挿入する	496
strmfe	ファイル名の拡張子を置き換える	499
strmfnc	ファイル名を作成する	500
strmfp	ファイル名を作成する	500
stcgfe	ファイル名から拡張子を取り出す	484
strbpl	文字列からポインタ配列を作成する	486
stcgfn	ファイル名からノードを取り出す	484
strsrt	文字列を ASCII コード順にソートする	510

時 間

関数名	機 能	ページ
asctime	時間情報を文字列に変換する	108
ctime	時間を int 型整数値から文字列に変換する	207
clock	プロセスの使用時間を求める	194
difftime	時間の引算を行う	208
ftime	システム時間を調べる	271
gmtime	時間を整数値から構造体に変換する	296
localtime	時刻を地方時に変換する	334
mktime	ローカル時間をカレンダー時間に変換する	375
strftime	時間情報を書式にしたがって文字列に変換する	492
time	システム時間を調べる	532
tzset	環境時間変数から外部時間変数を設定する	540

日本語操作

関数名	機 能	ページ
iskana	カナ文字またはカナ句読点の判定を行う	318
iskpun	カナ句読点の判定を行う	318

iskmoji	カナ文字の判定を行う	318
isalkana	英文字またはカナ文字の判定を行う	314
ispnkana	英句読点またはカナ句読点の判定を行う	314
isalnmkana	英数字またはカナ文字の判定を行う	314
isprkana	空白を含む表示可能文字の判定を行う	314
isgrkana	空白を除く表示可能文字の判定を行う	314
iskanji	シフト JIS コードの 1 バイト目の判定を行う	319
ikanji2	シフト JIS コードの 2 バイト目の判定を行う	319
jstrncmp	漢字を含む 2 つの文字列を比較する	325
jstrchr	漢字を含む指定文字の最初の位置を調べる	323
jstrrchr	漢字を含む指定文字の最後の位置を調べる	326
jstrcmp	漢字を含む 2 つの文字列を比較する	324
jiszen	全角文字の判定を行う	322
jisl0	全角空白文字または JIS 第一水準記号の判定を行う	322
jisl1	JIS 第一水準漢字の判定を行う	322
jisl2	JIS 第二水準漢字の判定を行う	322
jisalpha	全角英文字の判定を行う	321
jisupper	全角英大文字の判定を行う	321
jislower	全角英小文字の判定を行う	321
jisdigit	全角数字の判定を行う	321
jiskata	全角カタカナ文字の判定を行う	321
jishira	全角ひらがな文字の判定を行う	321
jiskigou	全角句読点の判定を行う	321
jisspace	全角空白文字の判定を行う	321
hantozen	半角文字を全角文字に変換する	298
zentohan	全角文字を半角文字に変換する	564

BASIC 関連

関数名	機能	ページ
a_cont	中断された PCM の動作を再開する	99
a_end	PCM の動作を強制終了する	101
a_play	PCM 録音を再生する	103
a_rec	PCM 録音を行う	106
a_stat	PCM の動作を調べる	111
a_stop	PCM の動作を中断する	113
asc	文字列の先頭の文字コードを求める	105

b_binS	整数を2進数の文字列に変換する	118
b_chrS	文字コードを文字に変換する	119
fix	浮動小数点値の整数部を求める	247
sgn	数値の符号を調べる	457
b_hexS	整数を16進数の文字列に変換する	145
b_itoa	整数を文字列に変換する	152
b_octS	整数を8進数の文字列に変換する	158
b_int	浮動小数点数を整数に変換する	150
val	文字列を数値に変換する	553
dskf	ディスクの空き容量を求める	211
b_fclose	ファイルをクローズする	125
b_fcloseall	すべてのファイルをクローズする	126
b_feof	指定ファイルがEOFかどうか調べる	127
b_fgetc	ファイルから1文字読み込む	128
b_fopen	ファイルをオープンする	130
b_fputc	ファイルへ1文字書き込む	132
b_fread	ファイルからデータを読み込む	133
b_freads	ファイルからデータを読み込む	134
b_fseek	ファイルポインタを移動する	136
b_fwrite	ファイルへデータを書き込む	137
b_fwrites	ファイルへデータを書き込む	138
randomize	rndの初期化を行う	427
rnd	浮動小数点数値の疑似乱数を発生させる	440
instr	特定の文字の並びを探し出す	313
b_midS	文字列を切り出す	156
b_leftS	文字列を切り出す	154
b_rightS	文字列を切り出す	162
b_mirrorS	文字列の並びを反転する	157
b_strchr	指定文字の最初の位置を調べる	170
b_strrchr	指定文字の最後の位置を調べる	170
b_stringS	1文字を複数個並べた文字列を作成する	175
b_strtok	トークンを探す	170
b_dateS	日付を文字列に変換する	121
b_timeS	時刻を文字列に変換する	176
b_dayS	曜日を求める	122
b_free	フリーエリアの大きさを調べる	135
b_inkeyS	キー入力を持ち読みとった1文字を求める	148

b_inkey0	キー入力を待たず読みとった1文字を求める	147
b_striS	整数値を文字列に変換する	174
b_strfS	浮動小数点数値を文字列に変換する	173
csrlin	カーソル位置のY座標を求める	206
pos	カーソル位置のX座標を求める	404
b_spaceS	スペースだけの文字列を作成する	167
using	スタック上のデータの型を判定する	548
beep	ベルを鳴らす	123
cls	テキスト画面を消去する	196
child	子プロセスをロードし実行する	187
b_exit	子プロセスを終了して親プロセスに戻る	124
width	画面の1行文字数を設定する	559
color	テキスト画面の文字属性を設定する	197
console	スクロール行範囲を設定する	198
b_input	フォーマットデータを入力する	149
b_linut	文字列を入力する	155
locate	カーソルを移動する	335
b_sprint	文字列をテキスト画面に表示する	168
b_iprint	数値をテキスト画面に表示する	151
b_fprint	数値をテキスト画面に表示する	131
b_tprint	テキスト画面上でタブスキップを行う	178
b_slprint	文字列をプリンタに出力する	129
b_ilprint	数値をプリンタに出力する	129
b_flprint	数値をプリンタに出力する	129
b_tlprint	プリンタにタブスキップを出力する	129
key	ファンクションキーに文字列を設定する	328
screen	表示画面の属性を設定する	447
pi	円周率を求める	402
b_pi	円周率の倍数を求める	161
b_setdate	日付を設定する	165
b_settime	時間を設定する	166
b_csw	カーソルスイッチをON/OFFする	120
b_tpalet	テキストのパレット色を設定する	177
b_strcmp	2つの文字列を比較する	171
b_strncpy	文字列をコピーする	171
b_stradd	2つの文字列を連結する	169
b_init	BASICの初期化を実行する	146

frename	ファイル名を変更する	260
fdelete	ファイルを削除する	232
apage	グラフィックのアクティブページを指定する	102
box	ボックスを描く	159
circle	円弧を描く	191
contrast	テキスト画面のコントラストを設定する	199
fill	ボックス描きパレット色で塗る	246
get	グラフィック画面の画像データを配列にとり込む	278
home	座標をホームポジションに設定する	299
hsv	色コードを求める	300
line	線を引く	332
paint	設定領域をパレット色で塗る	399
palet	パレットして色コードを割りあてる	400
point	グラフィック画面の座標のパレットコードを求める	403
pset	グラフィック画面の1点に色を塗る	411
put	グラフィック画面に画像データを表示する	412
rgb	RGBの色コードを調べる	437
symbol	グラフィック画面に文字列を表示する	524
vpage	グラフィック画面の各ページの表示制御を行う	557
window	グラフィック画面のクリッピング領域の座標を設定する	560
wipe	グラフィック画面を消去する	561
img_set	画像とり込みの設定を行う	311
img_still	画像とり込みの設定を行う	312
img_pos	画像とり込み時の画面水平位置を補正する	307
img_color	とり込み画像の色調補正を行う	303
v_cut	スーパーインポーズ時のテレビ画面表示を制御する	554
crt	テレビ画面とビデオ画面を切り替える	204
img_ht	グラフィック画面とテレビ画面の半透明制を行う	305
img_save	グラフィック画面のデータをファイルに書き込む	309
img_load	グラフィック画面のデータをファイルから読み込む	306
img_scrn	グラフィック画面のモード設定を行う	310
img_put	グラフィック画面のデータを配列にとり込む	308
keysns	キー入力の有無を判定する	329
img_home	グラフィック画面の表示位置を移動する	304
mouse	マウスの制御を行う	377
msarea	マウスカーソルの移動範囲を設定する	380
msbtn	マウスボタンの指定状態までの時間を調べる	382

m_spos	マウスカーソル座標を変数に設定する	384
m_stat	マウスの状態を調べる	385
setm_spos	マウスカーソルの位置を設定する	455
m_alloc	トラックバッファを確保しクリアする	343
m_assign	FM 音源のチャンネルにトラックを割りあてる	345
m_cont	一時中断していた演奏を再開する	349
m_free	トラックの残り容量を求める	368
m_init	FM 音源を初期化する	370
m_play	指定チャンネルに割りあてたトラックデータを演奏する	378
m_stat	チャンネルの状態を初期化する	387
m_stop	指定チャンネルの演奏を一時中断する	388
m_tempo	テンポを指定する	389
m_trk	MML による楽譜データをトラックに書き込む	390
m_vget	音色データを配列にとり込む	391
m_vset	音色データを指定音色番号のバッファに登録する	392
md_cont	MIDI や FM 音源の指定チャンネルの演奏の再開を行う	350
md_init	MIDI ボードの初期化を行う	351
md_off	MIDI ボードへの出力を停止する	352
md_on	MIDI ボードへの出力を開始する	353
md_play	MIDI や FM 音源の指定チャンネルの演奏を開始する	354
md_regr	MIDI コントローラのレジスタを読み出す	355
md_regw	MIDI コントローラもレジスタにデータを書き込む	356
md_stat	MIDI ボードへの出力状態を調べる	357
md_stop	MIDI や FM 音源の指定チャンネルの演奏を一時中断する	358
md_wrt	MIDI ボードにデータを出力する	359
sp_clr	スプライトをクリアする	465
sp_color	スプライトのパレット色を変更する	466
sp_def	スプライトのパターンデータを定義する	467
sp_disp	スプライトの表示制御を行う	468
sp_move	スプライトプレーンの表示位置を設定する	470
sp_off	スプライト表示を OFF にする	472
sp_on	スプライト表示を ON にする	473
sp_pat	スプライトのパターンデータを配列にとり込む	474
sp_init	スプライトパターンを初期化する	469
sp_set	スプライトプレーンデータをセットする	477
sp_stat	指定スプライトの状態を調べる	478
bg_scroll	バックグラウンドのスクロール座標を指定する	142

bg_set	バックグラウンドの設定を行う	143
bg_stat	指定バックグラウンドの状態を調べる	144
bg_fill	バックグラウンドテキストをデータで埋める	139
bg_put	バックグラウンドテキストに書き込む	141
bg_get	バックグラウンドテキストを読み込む	140
stick	ジョイスティックの状態を調べる	485
strig	指定ジョイスティックのボタン状態を調べる	495

IOCS コール

関数名	機能	ページ
B_KEYINP	キーデータの読み出しを行う	62
B_KEYSNS	キーデータバッファ状態を調べる	63
B_SFTSNS	シフトキーの押下状態を調べる	83
BITSNS	キーの押下状態を調べる	61
SKEYSET	入力されたキーコードを設定する	357
TVCTRL	テレビコントロールコードを入力しテレビを操作する	394
LEDMOD	LED キーのモードを設定する	274
TGUSEMD	画面のモードを制御する	379
DEFCHR	指定漢字パターンを外字エリアに書き込む	152
CRTMOD	CRT のモードを設定する	138
CONTRAST	コントラストを設定する	131
HSVTORGB	HSV 方式から RGB 方式に切り替える	233
TRAP15	レジスタにデータをセットし trap # 15 命令を実行する	392
TPALET	テキストのパレットを指定する	390
TPALET2	テキストのパレットを指定する	391
TCOLOR	テキストのカラーを指定する	376
FNTGET	指定漢字パターンを指定アドレスへ読み込む	197
TEXTGET	指定ドット座標からパターンを指定アドレスへ読み込む	377
TEXTPUT	指定ドット座標からパターンを指定アドレスへ書き込む	378
CLIPPUT	指定クリップ座標に従いクリッピング処理を行う	115
SCROLL	テキスト/グラフィックの表示座標の設定と調整を行う	342
B_CURON	カーソルを ON にする	35
B_CUROFF	カーソルを OFF にする	34
B_PUTC	指定した 1 バイトデータを表示する	74
B_PRINT	指定した文字列データを表示する	73
B_COLOR	カラー属性を設定する	32

B_LOCATE	カーソルを指定位置に設定する	65
B_DOWN_S	1行下へスクロールする	38
B_UP_S	1行上にスクロールする	86
B_DOWN	n行下へスクロールする	37
B_UP	n行下へスクロールする	85
B_RIGHT	n桁右へスクロールする	81
B_LEFT	n桁左へスクロールする	64
B_CLR_ED	カーソル位置から最終行右端まで画面を消去する	30
B_CLR_ST	先頭行左端からカーソル位置まで画面を消去する	31
B_CLR_AL	画面全体を消去する	29
B_ERA_ED	カーソル位置から現在行右端まで消去する	44
B_ERA_ST	現在行左端からカーソル位置まで消去する	45
B_ERA_AL	現在行全体を消去する	43
B_INS	カーソル設定行に n 行挿入する	59
B_DEL	カーソル設定行から n 行削除する	36
B_CONSOL	表示範囲を指定する	33
B_PUTMES	カーソル位置に指定文字列を表示する	75
SET232C	RS-232C モードを設定する	346
LOF232C	RS-232C 受信バッファ内のデータ数を求める	280
INP232C	RS-232C 受信データを読み込む	238
ISNS232C	RS-232C 受信データをチェックする	257
OSNS232C	RS-232C 送信が可能かどうかチェックする	319
OUT232C	RS-232C 送信を行う	320
JOYGET	ジョイスティックのデータを読み込む	259
INIT_PRN	プリンタポートを初期化する	236
SNSPRN	プリンタ出力が可能かどうかチェックする	361
OUTLPT	プリンタ出力を行う	321
OUTPRN	プリンタ出力を行う	322
B_SEEK	指定トラックまでシークする	82
B_VERIFY	データの比較チェックを行う	89
B_READDI	診断のためのチェックを行う	77
B_DSKINI	ディスクの初期化を行う	41
B_DRVSNS	ディスクのステータスを調べる	40
B_WRITE	ディスクにデータを書き込む	92
B_READ	ディスクからデータを読み込む	76
B_RECALI	トラック 0 へシークする	80
B_WRITED	削除データを書き込む	93

B_BADFMT	不良トラックを使用不可能とする	26
B_READDL	削除データを読み込む	78
B_FORMAT	物理フォーマットを行う	46
B_DRVCHK	指定ドライブの状態チェックと状態設定を行う	39
B_EJECT	イジェクトを行う	42
BINDATEBCD	日付を時計の形式に変換する	54
BINDATESET	日付を時計に設定する	57
TIMEBCD	時間を時計の形式に変換する	382
TIMESET	時間を時計に設定する	389
BINDATEGET	時計から日付を読み込む	56
DATEBIN	日付をBCD表現から2進数に変換する	148
TIMEGET	時計から時間を読み込む	386
TIMEBIN	時間をBCD表現から2進数に変換する	384
DATECNV	日付を表す文字列を2進データに変換する	150
TIMECNV	時間を表す文字列を2進データに変換する	385
DATEASC	2進データを日付を表す文字列に変換する	146
TIMEASC	2進データを時間を表す文字列に変換する	381
DAYASC	2進データを曜日を表す文字列に変換する	151
ALARMMOD	アラームの禁止/許可を設定する	21
ALARMSET	アラームの時間と処理アドレスを設定する	22
ALARMGET	アラームの時間と処理アドレスを読み込む	20
ADPCMOUT	ADPCMヘデータを出力する	16
ADPCMINP	ADPCMからデータを入力する	9
ADPCMAOT	ADPCMヘアレイチェーンモードでデータを出力する	7
ADPCMAIN	ADPCMからアレイチェーンモードでデータを入力する	5
ADPCMLOT	ADPCMヘリンクアレイチェーンモードでデータを出力する	13
ADPCMLIN	ADPCMからリンクアレイチェーンモードでデータを入力する	11
ADPCMSNS	ADPCMの実行モードをセンスする	18
ADPCMMOD	ADPCMの実行を制御する	15
OPMSET	FM音源にデータを書き込む	315
OPMSNS	FM音源のステータスを読み込む	316
OPMINTST	FM音源ICによる割り込みを制御する	314
TIMERDST	MFPのTIMER-Dによる割り込みを制御する	388
VDISPST	V-DISPのカウンタの割り込みを制御する	403
CRTCRA	CRTCに設定した割り込みラスタによる割り込みを制御する	137
HSYNCST	H-SYNCの起動による割り込みを制御する	234
PRNINTST	プリンタの割り込みを制御する	328

MS_INIT	マウスの初期化を行う	295
MS_CURON	マウスカーソルを表示する	292
MS_CUROF	マウスカーソルを消す	291
MS_STAT	マウスカーソルの表示モードを調べる	304
MS_GETDT	マウスの移動量とボタンの ON/OFF を調べる	294
MS_CURGT	マウスカーソルの座標を調べる	290
MS_CURST	マウスカーソルの座標を設定する	293
MS_LIMIT	マウスカーソルの移動範囲を設定する	296
MS_OFFTM	マウスボタンを離すまでの時間を調べる	297
MS_ONTM	マウスボタンを押すまでの時間を調べる	298
MS_PATST	マウスカーソルのパターンを定義する	299
MS_SEL	マウスカーソルを選択する	301
MS_SEL2	マウスカーソルを複数個選択する	302
SKEY_MOD	ソフトキーボードの制御を行う	355
DENSNS	電卓のセンスを行う	154
ONTIME	BIOS が起動してからの総時間を調べる	310
B_INTVCS	ベクタの設定を行う	60
B_SUPER	スーパーバイザモードへの切り替えを行う	84
B_BPEEK	指定アドレスからデータをバイト単位で読み込む	27
B_WPEEK	指定アドレスからデータをワード単位で読み込む	90
B_LPEEK	指定アドレスからデータをロングワード単位で読み込む	66
B_MEMSTR	指定アドレスからバイト数を指定してデータを読み込む	69
B_BPOKE	指定アドレスへデータをバイト単位で書き込む	28
B_WPOKE	指定アドレスへデータをワード単位で書き込む	91
B_LPOKE	指定アドレスへデータをロングワード単位で書き込む	67
B_MEMSET	指定アドレスへバイト数を指定してデータを書き込む	68
DMAMOVE	DMA 転送を行う	166
DMAMOV_A	アレイチェーンモードで DMA 転送を行う	164
DMAMOV_L	リンクアレイチェーンモードで DMA 転送を行う	168
DMAMODE	DMA の実行モードをセンスする	163
BOOTINF	ブート情報を求める	70
ROMVER	ROM のバージョンと作成年月日を調べる	341
G_CLR_ON	グラフィック画面を消去して表示モードにする	201
GPALET	グラフィックのパレットを設定する	218
SFTJIS	シフト JIS コードを JIS 漢字コードに変換する	354
JISSFT	JIS 漢字コードをシフト JIS コードに変換する	258
AKCONV	ANK コードをシフト JIS コードに変換する	19

RMACNV	ローマ字変換を行う	339
DAKJOB	全角文字の濁点処理を行う	145
HANJOB	全角文字の半濁点処理を行う	219
OS_CURON	カーソルを表示する	318
OS_CUROF	カーソルを消す	317
APAGE	グラフィック画面の書き込みページを設定する	25
VPAGE	グラフィック画面の表示ページを設定する	407
HOME	グラフィック画面の表示位置を設定する	232
WINDOW	グラフィック画面にウィンドウを設定する	409
WIPE	グラフィック画面を消去する	410
PSET	グラフィック画面に点を表示する	331
POINT	グラフィック画面上の点のパレットコードを求める	326
LINE	グラフィック画面に線を引く	275
BOX	グラフィック画面にボックスを描く	72
FILL	グラフィック画面にボックスを描き中を塗る	192
CIRCLE	グラフィック画面に円を描く	113
PAINT	グラフィック画面をペイントする	323
SYMBOL	グラフィック画面に文字を書く	375
GETGRM	グラフィック画面の読み込みを行う	209
PUTGRM	グラフィック画面の書き込みを行う	335
SP_INIT	スプライト面の初期化を行う	366
SP_ON	スプライト面の表示を ON にする	368
SP_OFF	スプライト面の表示を OFF にする	367
SP_CGCLR	PCG をクリアする	363
SP_DEFCG	PCG を設定する	364
SP_GTPCG	PCG の読み出しを行う	365
SP_REGST	スプライトレジスタの設定を行う	370
SP_REGGT	スプライトレジスタの読み出しを行う	369
BGSCRLST	BG スクロールレジスタの設定を行う	50
BGSCRLGT	BG スクロールレジスタの読み出しを行う	49
BGCTRLST	BG コントロールレジスタの設定を行う	48
BGCTRLGT	BG コントロールレジスタの読み出しを行う	47
BGTEXTCL	BG テキストを 1 つのコードで埋める	51
BGTEXTST	BG テキストを設定する	53
BGTEXTGT	BG テキストの読み出しを行う	52
SPALET	スプライトパレットの設定と読み出しを行う	362
TXXLINE	テキスト画面に X 方向ラインを引く	400

TTYLINE	テキスト画面に Y 方向ラインを引く	401
TXBOX	テキスト画面にボックスを描く	396
TXFILL	テキスト画面にボックスを描き中を塗る	397
TXREV	テキスト画面を反転表示する	399
TXRASCPY	テキスト画面のラスタコピーを行う	398
ABORTRST	アボートフラグをリセットする	4
IPLERR	IPL 時のエラー処理を行う	256
ABORTJOB	アボート処理を行う	3

DOS コール

関数名	機能	ページ
EXIT	プログラムを終了する	180
GETCHAR	キーボードより文字コードを入力して表示する	204
PUTCHAR	文字コードを表示する	334
COMINP	RS-232C 回線より 1 文字入力する	118
COMOUT	RS-232C 回線に 1 文字出力する	129
PRNOUT	プリンタへ 1 文字出力する	329
INPOUT	キー入力/センスを行う	239
INKEY	キーコードを入力する	237
GETC	キーコードを入力する	203
PRINT	文字列を表示する	327
GETS	文字列を入力する	214
KEYSNS	標準入力の入力状態を調べる	261
KFLUSHGP	バッファを空にしてキーボード入力を行う	263
KFLUSHIO	バッファを空にしてキーボード入力を行う	266
KFLUSHIN	バッファを空にしてキーボード入力を行う	265
KFLUSHGC	バッファを空にしてキーボード入力を行う	262
KFLUSHGS	バッファを空にしてキーボード入力を行う	264
FFLUSH	ディスクをリセットする	185
CHGDRV	カレントドライブを指定する	109
DRVCTRL	指定ドライブの状態のチェックと設定を行う	170
CONSNS	画面出力が可能かどうか調べる	130
PRNSNS	プリンタ出力が可能かどうか調べる	330
CINSNS	RS-232C 回線からの入力が可能かどうか調べる	112
COUTSNS	RS-232C 回線への出力が可能かどうか調べる	132
FATCHK	指定ファイルのドライブ番号とセクタの関連を調べる	182

FATCHK2	大容量ドライブのファイルのセクタの連続性を調べる	183
HENDSPMO	漢字変換行制御・モード表示ウィンドウのオープン	225
HENDSPMP	漢字変換行制御・モード表示ウィンドウへのノーマル表示	226
HENDSPMR	漢字変換行制御・モード表示ウィンドウへのリバース表示	227
HENDSPMC	漢字変換行制御・モード表示ウィンドウのクローズ	224
HENDSPIO	漢字変換行制御・入力ウィンドウのオープン	221
HENDSPIP	漢字変換行制御・入力ウィンドウへのノーマル表示	222
HENDSPIR	漢字変換行制御・入力ウィンドウのリバース表示	223
HENDSPIC	漢字変換行制御・入力ウィンドウのクローズ	220
HENDSPSO	漢字変換行制御・候補ウィンドウのオープン	229
HENDSPSP	漢字変換行制御・候補ウィンドウへのノーマル表示	230
HENDSPSR	漢字変換行制御・候補ウィンドウへのリバース表示	231
HENDSPSC	漢字変換行制御・候補ウィンドウのクローズ	228
CURDRV	カレントドライブの番号を求める	142
GETSS	リターンキーまでの文字列を入力する	215
FGETC	指定ファイルハンドルから1文字入力を行う	186
FGETS	指定ファイルハンドルから文字列入力を行う	187
FPUTC	指定文字コードをファイルハンドルへ出力する	199
FPUTS	指定文字列ファイルハンドルへ出力する	200
ALLCLOSE	ファイルハンドルをクローズする	24
SUPER	スーパーバイザモードとユーザーモードの切り替えを行う	372
FNCKEYGT	再定義可能キーの読み出しを行う	193
FNCKEYST	再定義可能キーの設定を行う	195
C_PUTC	コンソール直接出力・1バイトデータの表示	134
C_PRINT	コンソール直接出力・文字列データの表示	133
C_COLOR	コンソール直接出力・属性の設定	97
C_LOCATE	コンソール直接出力・カーソル位置の設定	116
C_DOWN_S	コンソール直接出力・1行下スクロール	102
C_UP_S	コンソール直接出力・1行上スクロール	140
C_DOWN	コンソール直接出力・n行下スクロール	101
C_UP	コンソール直接出力・n行上スクロール	139
C_RIGHT	コンソール直接出力・n桁右スクロール	136
C_LEFT	コンソール直接出力・n桁左スクロール	114
C_CLS_ED	コンソール直接出力・カーソル位置から最終行右端まで消去	95
C_CLS_ST	コンソール直接出力・先頭行左端からカーソル位置まで消去	96
C_CLS_AL	コンソール直接出力・画面消去	94
C_ERA_ED	コンソール直接出力・カーソル位置から行右端まで消去	104

C_ERA_ST	コンソール直接出力・行左端からカーソル位置まで消去	105
C_ERA_AL	コンソール直接出力・カーソル行消去	103
C_INS	コンソール直接出力・カーソル行に n 行挿入	111
C_DEL	コンソール直接出力・カーソル行から n 行削除	100
C_FNKMOD	コンソール直接出力・ファンクションキー行のモード設定	106
C_WINDOW	コンソール直接出力・スクロール範囲の設定	144
C_WIDTH	コンソール直接出力・画面サイズの設定	143
C_CURON	コンソール直接出力・カーソル表示モードの設定	99
C_CUROFF	コンソール直接出力・カーソル非表示モードの設定	98
K_KEYINP	コンソール直接入力・キーボード入力	271
K_KEYSNS	コンソール直接入力・キーコードの先読み	272
K_SFTSNS	コンソール直接入力・シフトキー状態のセンス	273
K_KEYBIT	コンソール直接入力・キーコードグループの状態通知	269
K_INSMOD	コンソール直接入力・挿入モードの ON/OFF 設定	268
INTVCS	ベクタにアドレスを設定する	241
PSPSET	プロセス管理情報を書き込む	332
GETTIM2	現在時刻を求める	216
SETTIM2	時刻を設定する	352
NAMESTS	ファイル名情報を求める	306
GETDATE	現在日付を求める	205
SETDATE	日付を設定する	349
GETTIME	現在時刻を求める	217
SETTIME	時刻を設定する	353
VERIFY	ベリファイラグの設定を行う	404
DUP0	ファイルハンドルの強制複写を行う	175
VERNUM	Human68k のバージョン番号を求める	406
KEEPPR	プログラムを常駐終了する	260
GETDPB	指定ドライブのドライブパラメータブロックを求める	206
BREAKCK	ブレークチェックの設定を行う	79
DRVXCHG	ドライブの入れ替えを行う	172
INTVCG	ベクタの値を求める	240
DSKFRE	ディスクの残り容量を求める	173
NAMECK	ファイル名をバッファに展開する	305
MKDIR	サブディレクトリを作成する	287
RMDIR	サブディレクトリを削除する	340
CHDIR	カレントディレクトリを変更する	108
CREATE	ファイルを新規に作成する	135

OPEN	ファイルをオープンする	311
CLOSE	ファイルをクローズする	117
READ	ファイルから読み込む	337
WRITE	ファイルへ書き込む	411
DELETE	ファイルを削除する	153
SEEK	ファイルポインタを移動する	343
CHMOD	ファイル属性を変更する	110
IOCTRLGT	デバイスドライバ直接入出力・装置情報の通知	246
IOCTRLST	デバイスドライバ直接入出力・装置情報の設定	253
IOCTRLRH	デバイスドライバ直接入出力・データの読み込み	251
IOCTRLWH	デバイスドライバ直接入出力・データの書き出し	255
IOCTRLRD	デバイスドライバ直接入出力・装置番号からの読み込み	250
IOCTRLWD	デバイスドライバ直接入出力・装置番号からの書き出し	254
IOCTRLIS	デバイスドライバ直接入出力・入力ステータスの通知	248
IOCTRLOS	デバイスドライバ直接入出力・出力ステータスの通知	249
IOCTRLDVCTL	デバイスドライバによる特殊コントロールを行う	242
IOCTRLDVGT	指定ドライブがローカルカリモートかを調べる	243
IOCTRLFDCTL	デバイスドライバによる特殊コントロールを行う	244
IOCTRLFDGT	指定ファイルがローカルカリモートかを調べる	245
IOCTRLRTSET	デバイスドライバに対する命令のリトライ数を設定する	252
DUP	ファイルハンドルを複写する	174
DUP2	ファイルハンドルを強制複写する	176
CURDIR	ドライブパス名を設定する	141
MALLOC	メモリ領域を確保する	284
MALLOC2	指定バイト数分のメモリを確保する	285
MFREE	メモリ領域を解放する	286
SETBLOCK	メモリブロックサイズを変更する	348
LOADEXEC	プログラムをロードして実行する	277
LOAD	プログラムをロードする	276
PATHCHK	プログラムロードのための準備作業を行う	324
LOADONLY	プログラムをロードする	278
EXECONLY	プログラムを実行する	179
BINDNO	ファイル名からモジュール番号を調べる	58
EXEC2	プログラムのロード/実行/モジュール番号の獲得を行う	177
EXIT2	プログラムを終了する	181
WAIT	プロセスの終了コードを求める	408
FILES	属性にしたがってファイルを検索する	190

NFILES	属性に従って次のファイルを検索する	309
SETPDB	指定プロセスに制御をわたす	351
GETPDB	現在のプロセスを表す pdbadr を求める	210
SETENV	環境変数を設定する	350
GETENV	環境変数の内容を取り出す	208
VERIFYFG	ベリファイフラグの設定状況を調べる	405
COMMON_CK	common 領域にブロックが存在するか調べる	119
COMMON_DEL	common 領域のブロックを消去する	120
COMMON_FRE	common 領域のデータのロックを解除する	121
COMMON_LK	common 領域のデータをロックする	123
COMMON_RD	common 領域のデータを読み出す	125
COMMON_WT	common 領域にデータを書き込む	127
MOVE	ファイル名を変更する	288
RENAME	ファイル名を変更する	338
FILEDATE	ファイルの日付/時間の読み出しと設定	188
MAKETMP	テンポラリファイルを作成する	282
NEWFILE	ファイルを新規に作成する	308
LOCK	ファイルのロックをする	279
UNLOCK	ファイルのロックを解除する	402
GETASSIGN	仮想ドライブや仮想ディレクトリの割り当て状況を調べる	202
MAKEASSIGN	仮想ドライブや仮想ディレクトリの割り当てを行う	281
RASSIGN	仮想ドライブ名や仮想ディレクトリ名をキャンセルする	336
S_MALLOC	指定バイト分のメモリをメインのメモリ管理から確保する	359
S_MFREE	メインのメモリ管理から確保した領域を開放する	360
S_PROCESS	サブのメモリ管理領域を設定する	371
DISKRED	ブロックデバイスの直接リード	155
DISKRED2	大容量のブロックデバイスから直抗データを読み込む	157
DISKWRT	ブロックデバイスの直接ライト	159
DISKWRT2	大容量のブロックデバイスに直抗データを書き込む	161
INDOSFLG	OS のワークエリア INDOS_FLG のアドレスを求める	235
SUPER_JSR	スーパーバイラ領域のプログラムをサブルーチンコールする	373
BUS_ERR	バスエラー発生のある可能性がある領域が読み書き可能か調べる	87
KILL_PR	自分自身のプロセスを削除する	267
TIME_PR	現在のタイマーのカウンタ値 (ミリ秒単位) を調べる	387
OPEN_PR	バックグラウンドタスクを登録する	312
GET_PR	指定 ID のバックグラウンドタスクの管理情報を調べる	212
SUSPEND_PR	スレッドを強制的に SLEEP 状態にする	374

SLEEP_PR	スレッドを SLEEP 状態にする	358
SEND_PR	スレッドにコマンドやデータを送る	344
CHANGE_PR	バックグラウンドタスクの実効権を破棄する	107

その他

関数名	機能	ページ
assert	論理エラーをテストする	110
getenv	環境変数の値を求める	286
longjmp	セーブされたスタック環境をリストアする	337
offsetof	構造体メンバの位置を求める	395
perror	エラーメッセージを表示する	401
putenv	環境変数の値を修正または追加する	417
abs	int 型整数値の絶対値を求める	96
labs	int 型整数の絶対値を求める	96
max	2つの引数のうち大きいものを求める	348
min	2つの引数のうち小さいものを求める	369
wabs	short 型整数値の絶対値を求める	96
rand	擬似乱数を発生する	426
setjmp	スタック環境をセーブする	451
srand	擬似乱数を初期化する	480
strerror	エラー番号に対応するエラーメッセージを調べる	491
va_arg	可変個の引数を値として読み込む	551
va_end	可変個の引数の参照を終了する	551
va_start	可変個の引数の参照を開始する	551



サトー株式会社

本社 〒545 大阪市阿倍野区長池町22番22号

電子機器事業本部 〒329-21 栃木県矢板市早川町174番地

液晶映像システム事業部 第2商品企画部

お問い合わせ先 〒162 東京都新宿区市谷八幡町8番地 電話 (03)3260-1161(大代表)

東京支社内 液晶映像システム事業部 第2商品企画部 ソフトウェア担当