

SHARP

SHARP  
COMPUTER  
SOFTWARE

△▽68000用

COMPILER  ver2.0  
PRO-68K

CライブラリマニュアルVOL.2



 68000用

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

Cライブラリマニュアル VOL.2

**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、大文字関数リファレンスに関して記述しています。

小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.1」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、「VOL.1 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 リファレンス (大文字)

---

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使用の手引きとして利用してください。

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 (大文字) 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。

---

## 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。

---

## 索引

---

50 音順、アルファベット順で整理しています。

---

## 関数索引

---

大文字関数、および小文字関数をアルファベット順で整理しています。

---

## 機能別関数索引

---

関数を分かりやすく機能別に分類しています。

# CONTENTS

## 第1章 リファレンス (大文字)

3

## 付 録

415

## 索 引

50 音順 ..... 421

アルファベット順 ..... 428

## 関数索引

432

## 機能別関数索引

445

# 第1章

## リファレンス (大文字)

# 第1章

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（大文字）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。

# ABORTJOB

レベル 0

**書 式** #include <iocslib.h>

void ABORTJOB ( );

**機 能** アボート処理を行います。

本ファンクションコールは Human68k が使用します。

**戻 り 値** 戻り値はありません。

リターンせずに、エラーアポートルーチン (OS のエラーが出たとき、中止 <A> を選ぶ) へ制御を移行します。

# ABORTRST

レベル 0

**書 式** #include <iocslib. h>

void ABORTRST ( ) ;

**機 能** アボートするためのフラグを、リセットします。  
本ファンクションコールは Human68k が使用します。

**戻 り 値** 戻り値はありません。

# ADPCMAIN

## レベル 0

**書 式** #include <iocslib.h>

```
void ADPCMAIN (TBLADDRESS, MODE, TBLCNT) ;
struct CHAIN *TBLADDRESS; /* チェーンテーブルのアドレス */
int MODE; /* モード */
int TBLCNT; /* チェーンテーブルの個数 */
struct CHAIN {
    int adr; /* 先頭アドレス */
    unsigned short len; /* 長さ */
};
```

### 機 能

ADPCM からデータを入力します。

データ長は 0x0001~0xFFFF です。

TBLADDRESS には入力データチェーンテーブルのアドレスを指定します。

データチェーンテーブルの内容は、先頭アドレス、長さ……先頭アドレス、長さの順番で定義しておきます。

MODE には、サンプリング周波数×256+出力モードを指定します。

#### ●サンプリング周波数

- 0 : 3.9KHz (1950 バイト/sec)
- 1 : 5.2KHz (2600 バイト/sec)
- 2 : 7.8KHz (3900 バイト/sec)
- 3 : 10.4KHz (5200 バイト/sec)
- 4 : 15.6KHz (7800 バイト/sec)

#### ●出力モード

- 0 : 音声出力カット
- 1 : 音声出力左
- 2 : 音声出力右
- 3 : 音声出力両方

TBLCNT には入力データチェーンテーブルの個数を指定します。

データ入力は一方向で入力しますが、モニタ出力を出力モードで指定できます。

**戻り値** 戻り値はありません。

プログラム例

```

#include      <stdlib.h>
#include      <iocslib.h>

#define SFREQ 1          /* サンプリング周波数(5.2KHz) */
#define SOUND 2         /* 入力モード(音声入力右) */
#define TSIZE 3         /* テーブルサイズ */
#define BSIZE 4096

static struct CHAIN  table[TSIZE]; /* chainテーブル */

main()
{
    int      set_table(struct CHAIN *, int);
    int      mode;

    if (set_table(table, TSIZE) == TSIZE) {
        mode = SFREQ * 256 + SOUND;
        ADPCMMAIN(table, mode, TSIZE);
    }
}

static int      set_table(struct CHAIN *p, int n)
{
    int      i;

    for (i = 0; i < n; i++) {
        /* The member 'adr' is badly declared */
        if ((p->adr = (INT)malloc(BSIZE)) == NULL)
            break;
        p->len = BSIZE;
        ++p;
    }
    return (i);
}

```

## ADPCMAOT

## レベル 0

**書 式** #include <iocslib.h>

```
void ADPCMAOT (TBLADDRESS, MODE, TBLCNT) ;
struct CHAIN *TBLADDRESS ; /* チェーンテーブルのアドレス */
int MODE ; /* モード */
int TBLCNT ; /* チェーンテーブルの個数 */
struct CHAIN {
    int adr ; /* 先頭アドレス */
    unsigned short ; /* 長さ */
} ;
```

**機 能**

ADPCM ヘデータを出力します。  
データ長は、0x0001~0xFFFF です。  
TBLADDRESS には、出力データチェーンテーブルのアドレスを指定します。  
アドレスの内容は、先頭アドレス、長さ……先頭アドレス、長さの順番で定義しておきます。

MODE には、サンプリング周波数×256+出力モードを指定します。

## ●サンプリング周波数

- 0 : 3.9KHz (1950 バイト/sec)
- 1 : 5.2KHz (2600 バイト/sec)
- 2 : 7.8KHz (3900 バイト/sec)
- 3 : 10.4KHz (5200 バイト/sec)
- 4 : 15.6KHz (7800 バイト/sec)

## ●出力モード

- 0 : 音声出力カット
- 1 : 音声出力左
- 2 : 音声出力右
- 3 : 音声出力両方

TBLCNT には、出力データチェーンテーブルの個数を指定します。

**戻 り 値**

戻り値はありません。

プログラム例

```

#include      <stdlib.h>
#include      <iocslib.h>

#define SFREQ 1          /* サンプリング周波数(5.2KHz) */
#define SOUND 1         /* 出力モード(音声出力左) */
#define TSIZE 3         /* テーブルサイズ */
#define BSIZE 4096

static struct CHAIN  table[TSIZE]; /* chainテーブル */

main()
{
    int      set_table(struct CHAIN *, int);
    int      mode;

    if (set_table(table, TSIZE) == TSIZE) {
        mode = SFREQ * 256 + SOUND;
        ADPCMAIN(table, mode, TSIZE);
        ADPCMAOT(table, mode, TSIZE);
    }

    static int      set_table(struct CHAIN *p, int n)
    {
        int      i;

        for (i = 0; i < n; i++) {
            /* The member 'adr' is badly declared */
            if ((p->adr = (INT)malloc(BSIZE)) == NULL)
                break;
            p->len = BSIZE;
            ++p;
        }
        return (i);
    }
}

```

# ADPCMINTP

## レベル 0

**書 式** #include <iocplib.h>

```
void ADPCMINTP (ADDRESS, MODE, LENGTH) ;
unsigned char *ADDRESS; /* データ読み込み領域のアドレス */
int MODE; /* モード */
int LENGTH; /* 入力データ長 */
```

**機 能** ADPCM からデータを入力します。

データ長が 0xFF00 以上の場合、入力処理は 0xFF00 単位で行われるので、リターンまでに時間がかかります。

ADDRESS には、データ読み込み領域のアドレスを指定します。

MODE には、サンプリング周波数×256+出力モードを指定します。

●サンプリング周波数

- 0 : 3.9KHz (1950 バイト/sec)
- 1 : 5.2KHz (2600 バイト/sec)
- 2 : 7.8KHz (3900 バイト/sec)
- 3 : 10.4KHz (5200 バイト/sec)
- 4 : 15.6KHz (7800 バイト/sec)

●出力モード

- 0 : 音声出力カット
- 1 : 音声出力左
- 2 : 音声出力右
- 3 : 音声出力両方

LENGTH には、入力データの長さを指定します。

データ入力はモノラルで入力しますが、モニタ出力を出力モードで指定できません。

**戻り値** 戻り値はありません。

プログラム例

プログラム例

```

#include      <iocslib.h>

#define SFREQ 1 /* サンプリング周波数(5.2KHz) */
#define SOUND 3 /* 入力モード(音声入力両方) */
#define DTLEN 8192 /* データ長 */

static char  dbuf[DTLEN]; /* データ読み込み領域 */

main()
{
    int      mode;
    mode = SFREQ * 256 + SOUND;
    ADPCMIMP(dbuf, mode, DTLEN);
}

```

## ADPCM LIN

## レベル 0

**書 式** #include <iocslib.h>

```
void ADPCM LIN (TBLADDRESS, MODE);
struct CHAIN2 * TBLADDRESS; /* アレイチェーンテーブルのアドレス
*/
int MODE; /* モード */
struct CHAIN2 {
    int adr; /* 先頭アドレス */
    unsigned short len; /* 長さ */
    struct CHAIN2 *next; /* 次のテーブルのアドレス */
};
```

**機 能**

ADPCM からデータを入力します。

TBLADDRESS には、入力データアレイチェーンテーブルの先頭アドレスを指定します。

TBLADDRESS が指す領域には先頭アドレス、長さ、次のテーブルアドレスが格納され、次のテーブルにチェーンしています。

最後のテーブルでは、次のテーブルアドレスには 0 を設定します。

MODE には、サンプリング周波数×256+出力モードを指定します。

## ●サンプリング周波数

- 0 : 3.9KHz (1950 バイト/sec)
- 1 : 5.2KHz (2600 バイト/sec)
- 2 : 7.8KHz (3900 バイト/sec)
- 3 : 10.4KHz (5200 バイト/sec)
- 4 : 15.6KHz (7800 バイト/sec)

## ●出力モード

- 0 : 音声カット
- 1 : 音声出力左
- 2 : 音声出力右
- 3 : 音声出力両方

データ入力は一チャンネルで入力しますが、モニタ出力を出力モードで指定できます。

**戻 り 値**

戻り値はありません。

プログラム例

```

#include      <stdlib.h>
#include      <iocslib.h>

#define SFREQ  0          /* サンプリング周波数(3.9KHz) */
#define SOUND  3          /* 出力モード(音声出力両方) */
#define TSIZE  3
#define BSIZE  4096

typedef struct CHAIN2  CH2;

main()
{
    CH2  *set_chain(int);
    CH2  *chain;          /* chainテーブル */
    int  mode;           /* モード */

    if ((chain = set_chain(TSIZE)) != NULL) {
        mode = SFREQ * 256 + SOUND;
        ADPCM LIN(chain, mode);
    }

static CH2  *set_chain(int n)
{
    CH2  *ip;
    CH2  *cp;
    CH2  *pp;
    int  i;

    for (i = 0; i < n; i++) {
        if ((cp = (CH2 *)malloc(sizeof(CH2))) == NULL)
            return (NULL);
        cp->len = BSIZE;
        /* The member 'adr' is badly declared */
        if ((cp->adr = (INT)malloc(BSIZE)) == NULL)
            return (NULL);
        if (i == 0)
            ip = cp;
        if (i >= 0)
            pp->next = cp;
        pp = cp;
        ++cp;
    }
    pp->next = NULL;
    return (ip);
}

```

# ADPCMLOT

## レベル 0

**書 式** #include <iocslib.h>

```
void ADPCMLOT (TBLADDRESS, MODE);
struct CHAIN2 * TBLADDRESS; /* アレイチェーンテーブルのアドレス */
int MODE; /* モード */
struct CHAIN2 {
    int adr; /* 先頭アドレス */
    unsigned short len; /* 長さ */
    struct CHAIN2 *next; /* 次のテーブルのアドレス */
};
```

**機 能**

ADPCM ヘデータを出力します。

TBLADDRESS には、出力データアレイチェーンテーブルの先頭アドレスを指定します。

TBLADDRESS が指す領域には、先頭アドレス、長さ、次のテーブルアドレスが格納され、次のテーブルにチェーンしています。

最後のテーブルでは、次のテーブルアドレスには 0 を設定します。

MODE には、サンプリング周波数×256+出力モードを指定します。

● サンプル周波数

- 0 : 3.9KHz (1950 バイト/sec)
- 1 : 5.2KHz (2600 バイト/sec)
- 2 : 7.8KHz (3900 バイト/sec)
- 3 : 10.4KHz (5200 バイト/sec)
- 4 : 15.6KHz (7800 バイト/sec)

● 出力モード

- 0 : 音声出力カット
- 1 : 音声出力左
- 2 : 音声出力右
- 3 : 音声出力両方

**戻り値**

戻り値はありません。

プログラム例

```

#include      <stdlib.h>
#include      <ioclib.h>

#define SOUND 3          /* 出力モード(音声出力両方) */
#define SFREQ 2         /* サンプルング周波数(7.8KHz) */
#define TSIZE 3
#define BSIZE 4096

typedef struct CHAIN2   CH2;

main()
{
    CH2 *set_chain(int);
    CH2 *chain;         /* chainテーブル */
    int mode;          /* モード */

    if ((chain = set_chain(TSIZE)) != NULL) {
        mode = SFREQ * 256 + SOUND;
        ADPCMLIN(chain, mode);
        ADPCMLOT(chain, mode);
    }

    static CH2 *set_chain(int n)
    {
        CH2 *ip;
        CH2 *cp;
        CH2 *pp;
        int i;

        for (i = 0; i < n; i++) {
            if ((cp = (CH2 *)malloc(sizeof(CH2))) == NULL)
                return (NULL);
            cp->len = BSIZE;
            /* The member 'adr' is badly declared */
            if ((cp->adr = (INT)malloc(BSIZE)) == NULL)
                return (NULL);
            if (i == 0)
                ip = cp;
            if (i >= 0)
                pp->next = cp;
            pp = cp;
            ++cp;
        }
        pp->next = NULL;
        return (ip);
    }
}

```

# ADPCM MOD

レベル 0

**書 式** #include <iocslib.h>

```
void ADPCMOUT (ADDRESS);
int MODE; /* モード */
```

**機 能** ADPCM の実行制御を行います。  
MODE には、次の動作モードを指定します。

- 0 : 終了
- 1 : 中止
- 2 : 再開

**戻り値** 戻り値はありません。

## プログラム例

```
#include <iocslib.h>

#define SFREQ 1 /* サンプリング周波数(5.2KHz) */
#define SOUND 3 /* 出力モード(音声出力両方) */
#define DTLEN 4096 /* データ長 */
#define EMODE 2 /* 実行モード(実行再開) */

static char dbuf[DTLEN]; /* 出力データ */

main()
{
    int mode;

    ADPCMOUT(EMODE);

    mode = SFREQ * 256 + SOUND;
    ADPCMOUT(dbuf, mode, DTLEN);
}
```

# ADPCMOUT

レベル 0

**書 式** #include <ioclib.h>

```
void ADPCMOUT (ADDRESS, MODE, LENGTH);  
unsigned char *ADDRESS; /* 出力データアドレス */  
int MODE; /* モード */  
int LENGTH; /* 出力データ長 */
```

**機 能**

ADPCM ヘーデータを出力します。

データ長が 0xFF00 以上の場合、出力処理は 0xFF00 単位で行われるので、リターンまでに時間がかかります。

ADDRESS には、出力データのアドレスを指定します。

MODE には、サンプリング周波数×256+出力モードを指定します。

●サンプリング周波数

- 0 : 3.9KHz (1950 バイト/sec)
- 1 : 5.2KHz (2600 バイト/sec)
- 2 : 7.8KHz (3900 バイト/sec)
- 3 : 10.4KHz (5200 バイト/sec)
- 4 : 15.6KHz (7800 バイト/sec)

●出力モード

- 0 : 音声出力カット
- 1 : 音声出力左
- 2 : 音声出力右
- 3 : 音声出力両方

LENGTH には、出力データ長を指定します。

**戻り値**

戻り値はありません。

## プログラム例

```

#include      <iocslib.h>

#define SFREQ 1          /* サンプリング周波数(5.2KHz) */
#define SOUND 3         /* 出力モード(音声出力両方) */
#define DTLEN 8192      /* データ長 */

static char  dbuf[DTLEN]; /* 出力データ */

main()
{
    int      mode;

    mode = SFREQ * 256 + SOUND;
    ADPCMOUT(dbuf, mode, DTLEN);
}

```

```

#include      <iocslib.h>

#define SFREQ 1          /* サンプリング周波数(5.2KHz) */
#define SOUND 3         /* 出力モード(音声出力両方) */
#define DTLEN 8192      /* データ長 */

static char  dbuf[DTLEN]; /* 出力データ */

main()
{
    int      mode;
    int      result;

    result = ADPCMSET();
    printf("ans = %d\n", result);
    if (result == 0) {
        mode = SFREQ * 256 + SOUND;
        ADPCMOUT(dbuf, mode, DTLEN);
    }
}

```

# ADPCMSNS

レベル 0

**書 式** #include <iocslib.h>

```
int ADPCMSNS ( );
```

**機 能** ADPCM の実行モードを調べます。

**戻 り 値** ADPCM の実行モードを返します。

0x00 : 何もしていない  
0x02 : 出力中 (ADPCMOUT 実行中)  
0x04 : 入力中 (ADPCMINP 実行中)  
0x12 : 出力中 (ADPCMAOT 実行中)  
0x14 : 入力中 (ADPCMAIN 実行中)  
0x22 : 出力中 (ADPCMLOT 実行中)  
0x24 : 入力中 (ADPCMLIN 実行中)

## プログラム例

```
#include <iocslib.h>

#define SFREQ 1 /* サンプリング周波数(5.2KHz) */
#define SOUND 3 /* 出力モード(音声出力両方) */
#define DTLEN 4096 /* データ長 */

static char dbuf[DTLEN]; /* 出力データ */

main()
{
    int mode;
    int result;

    result = ADPCMSNS();
    printf("sns = %d\n", result);
    if (result == 0) {
        mode = SFREQ * 256 + SOUND;
        ADPCMOUT(dbuf, mode, DTLEN);
    }
}
```

## AKCONV

## レベル 0

**書 式** #include <iocslib.h>

```
int AKCONV (MODE, CODE) ;
```

```
int CODE; /* ANK コード */
```

```
int MODE; /* 文字種 */
```

**機 能** ANK コードをシフト JIS コードに変換します。

CODE には ANK コードを指定します (0x20~0x7E、0xA1~0xDF)。

MODE には文字種を指定します。

0 : ひらがな

1 : カタカナ

**戻り値** シフト JIS コードを返します。

ただし、上位 16 ビットが 0xFFFF ならエラーが発生したことを示します。

```

#include <stdio.h>
#include <iocslib.h>

main()
{
    int time;
    int offtime;
    int job;

    /* アラームを設定する */
    ALARMMOD(1);
    ALARMDT(time, coltime, #job);

    printf("job = %d\n", job);
    printf("曜日 = %d\n", (time >> 24) & 0x07);
    printf("日 = %d\n", (time >> 16) & 0x07);
    printf("時 = %d\n", (time >> 8) & 0x07);
    printf("分 = %d\n", (time >> 0) & 0x07);
    printf("長さ(分) = %d\n", offtime);
}

```

# ALARMGET

レベル 0

## 書式

```
#include <ioclib.h>

int ALARMGET (DATETIME, OFFTIME, JOB) ;
int *DATETIME; /* アラームの時間を指すポインタ */
int *OFFTIME; /* オフされるまでの時間を指すポインタ */
int *JOB; /* 処理アドレス */
```

## 機能

アラームの時間と処理アドレスを読み込みます。

## 戻り値

処理アドレス (ALARMSET 参照) が返されます。

そして、アラームの時間と処理アドレスが、DATETIME, OFFTIME, JOB の各変数に格納されます。

DATETIME の形式と設定内容は次の通りです。

日・時・分はいずれも BCD2 桁です。

0000WWWW DDDDDDDD HHHHHHHH MMMMMMMM

WWWW : 曜日 (0~6) 0 : 日曜日

DDDDDDDD : 日 (01~31)

HHHHHHHH : 時 (00~23)

MMMMMMMM : 分 (00~59)

曜日が 0xF、日、時、分が 0xFF になっている場合は、無指定を表します。

OFFTIME には、テレビあるいはコンピュータが OFF されるまでの時間が設定されます (単位は分)。

JOB には処理アドレスが設定されます (ALARMSET 参照)。

## プログラム例

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

main()
{
    int time; /* 日付・時刻 */
    int offtime; /* offになるまでの時間 */
    int job; /* 処理アドレス */

    ALARMMOD(1); /* アラーム許可状態にする */

    ALARMGET(&time, &offtime, &job);

    printf(" job = 0x%x\n", job);
    printf(" 曜日 = %x\n", (time >> 24) & 0x0f);
    printf(" 日 = %x\n", (time >> 16) & 0xff);
    printf(" 時 = %x\n", (time >> 8) & 0xff);
    printf(" 分 = %x\n", (time >> 0) & 0xff);
    printf("長さ(分) = %x\n", offtime);
}
```

# ALARMMOD

レベル 0

**書 式** #include <iocslib.h>

```
int ALARMMOD (MODE) ;
int MODE; /* モード */
```

**機 能** アラームの禁止/許可の設定、および現在の状態の通知を行います。  
MODE には次の内容を指定できます。

- 0 : 禁止
- 1 : 許可
- 2 : 通知要求

**戻 り 値** 設定されたアラームの状態を返します。

- 0 : 禁止
- 1 : 許可

## プログラム例

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

#define M_UNABL 0 /* 禁止*/
#define M_ENABL 1 /* 許可 */
#define M_SENSE 2 /* 通知要求 */

main()
{
    if (ALARMMOD(M_SENSE) == M_UNABL) {
        printf("アラームは禁止されています\n");
        exit(1);
    }
}
```

# ALARMSET

レベル 0

**書 式** # include <iocslib. h>

```
int ALARMSET (DATETIME, OFFTIME, MODE);
int DATETIME; /* 日付・時間 */
int OFFTIME; /* オフまでの時間 */
int MODE; /* モード */
```

**機 能** アラームの時間と処理アドレスを設定します。  
DATETIME の形式と指定内容は次の通りです。  
日・時・分はすべて BCD2 桁です。

0000WWWW DDDDDDDD HHHHHHHH MMMMMMMM

WWWW : 曜日 (0~6) 0 : 日曜日  
DDDDDD : 日 (01~31)  
HHHHHH : 時 (00~23)  
MMMMMM : 分 (00~59)

曜日に 0xF、日、時、分に 0xFF を指定すると、それぞれが無指定になります。  
ただし、すべてを無指定にしないでください。

OFFTIME には、テレビまたはコンピュータが OFF されるまでの時間を設定します。

0 : いつまでも OFF にしない  
?? : OFF するまでの時間を 1 分単位で指定する

MODE には処理アドレスを指定します。

0 : パワー ON して、テレビオン&コンピュータモードにする。  
-1 : パワー ON のみ。テレビコントロールは行いません。  
1~0x3F : テレビコントロールを行います。(TVCTRL 参照)  
0x40~0xFFFFFFFF : 処理アドレス。

処理アドレスのプログラムの先頭は、0x60 でなければなりません。

**戻り値** エラーのとき -1 を返します。

## プログラム例

```
#include <iocslib.h>

#define YOBI 0x00 /* 曜日 (BCD) */
#define DATE 0x25 /* 日 (BCD) */
#define TIME 0x12 /* 時 (BCD) */
#define MINT 0x00 /* 分 (BCD) */

main()
{
    int dt; /* 日付・時刻 */
    int length; /* offになるまでの時間 */
    int mode; /* モード */
    int almmode; /* アラームモード */

    length = 10;
    mode = 0;
    almmode = 1;

    ALARMMOD(almmode); /* アラーム許可 */

    dt = YOBI << 24;
    dt |= DATE << 16;
    dt |= TIME << 8;
    dt |= MINT << 0;

    ALARMSET(dt, length, mode); /* アラーム設定 */
}
```

# ALLCLOSE

レベル 1

**書 式** # include <doslib. h>

```
void ALLCLOSE ( ) ;
```

**機 能** 現在オープンしているファイルハンドルを、すべてクローズします。  
子プロセスがオープンしたファイルハンドルについても同様です。  
ただし、該当プロセスの親プロセスがオープンしたファイルハンドルは、クローズされません。

**戻 り 値** 戻り値はありません。

注：他のレベルでファイルをオープンしている場合（fopenなどでファイルストリームをオープンしている時など）は、この関数を使用してはいけません。

# APAGE

レベル 0

**書 式** # include <iocslib.h>

```
int APAGE (MODE);
int MODE; /* 書き込みページ */
```

**機 能** グラフィック画面の書き込みページを設定します。  
書き込みページは MODE で指定します。

- 0 : 書き込みページに 0 ページを指定
- 1 : 書き込みページに 1 ページを指定
- 2 : 書き込みページに 2 ページを指定
- 3 : 書き込みページに 3 ページを指定
- 1 : 現在の書き込みページを調べる

**戻り値** 終了コードを返します。

- 0 : 正常終了
- 1 : グラフィックは使用不可
- 2 : ページ引数が規定外
- 3 : 指定されたページは、現在のモードでは設定不可

MODE が -1 のとき正常終了ならば 0~3 の値を返します。

# B\_BADFMT

レベル 0

**書 式** # include <ioclib. h>

```
int B_BADFMT (DRIVE, RECNO, MODE) ;
int DRIVE; /* ドライブ番号 */
int RECNO; /* レコード番号 */
int MODE; /* インタリーブコード */
```

**機 能** 不良トラックを使用不能にします (ハードディスクのみ)。  
DRIVE には、pda×256 を指定します。

- pda  
0x80~0x8F ハードディスク

RECNO には 256 バイト単位のレコード番号を指定します。

MODE にはインタリーブコード (6、1) を指定します (通常 10 メガバイトドライブは 6、20・40 メガバイトドライブは 1)。

**戻り値** 0xFFFFFFFF?? ならばエラーを表します。  
正常終了ならば正の数 (0 も) を返します。

# B\_BPEEK

レベル 0

**書式** #include <iocslib.h>

```
int B_BPEEK (ADDRESS);
/* unsigned char *ADDRESS; /* データ格納領域のアドレス */
```

**機能** ADDRESS で指定されたアドレスからデータをバイト単位で読み込みます。**戻り値** 読み込んだデータを返します。**プログラム例**

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

static char mem = 0x10;

main()
{
    char *addr; /* アドレス */
    int rdata;

    addr = &mem;
    rdata = B_BPEEK(addr);
    printf("0x%02x%fn", addr, rdata);
}
```

# B\_BPOKE

レベル 0

**書 式** # include <iocslib.h>

```
void B_BPOKE (ADDRESS, DATA) ;  
/* unsigned char * ADDRESS; /* データ格納領域のアドレス */  
   int DATA; /* 書き込むデータ */
```

**機 能** ADDRESS で指定されたアドレスへ、データを書き込みます (バイト単位)。

**戻 り 値** 戻り値はありません。

## プログラム例

```
#include <stdio.h>  
#include <iocslib.h>  
  
static unsigned char mem = 0x10;  
  
main()  
{  
    unsigned char *addr;  
  
    addr = &mem;  
    printf("x: 0x%02x\n", addr, mem);  
    B_BPOKE(addr, 0xff);  
    printf("x: 0x%02x\n", addr, mem);  
}
```

# B\_CLR\_AL

レベル 0

**書 式** # include <iocslib.h>

void B\_CLR\_AL ( );

**機 能** テキスト画面全体をクリアします。カーソルはホームポジションに移動します。

**戻り値** 戻り値はありません。



# B\_CLR\_ED

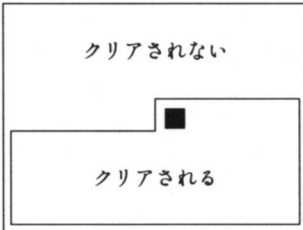
レベル 0

**書 式** # include <ioclib.h>

viod B\_CLR\_ED ( ) ;

**機 能** カーソル位置から最終行右端までテキスト画面をクリアします。

テキスト画面



**戻り値** 戻り値はありません。

# B\_CLR\_ST

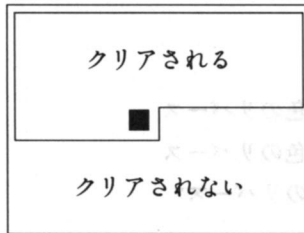
レベル 0

**書 式** # include <iocslib.h>

```
void B_CLR_ST ( );
```

**機 能** 先頭行左端からカーソル位置までテキスト画面をクリアします。

テキスト画面



**戻り値** 戻り値はありません。

# B\_COLOR

レベル 0

**書式** # include <iocslib.h>

```
int B_COLOR (COLOR) ;
int COLOR; /* カラー属性 */
```

**機能** COLORで指定した属性を設定します。

COLORに指定できる値とその内容は次の通りです。

- |           |                |
|-----------|----------------|
| 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、COLORに-1を指定したときは、属性の通知のみ行います。

**戻り値** 前の属性を返します。

-1が返ったときは、有効範囲外を指定したことを示します。

# B\_CONSOL

レベル 0

**書 式** # include <iocslib. h>

```
int B_CONSOL (XS, YS, XL, YL) ;
int XS; /* 先頭 X 座標 */
int YS; /* 先頭 Y 座標 */
int XL; /* X の桁数-1 */
int YL; /* Y の行数-1 */
```

**機 能**

表示範囲を指定します。

カーソルはホームポジションに移動します。

各座標の指定方法は次の通りです。

XS : 先頭 X 座標 (16 の倍数を指定。0~1008)

YS : 先頭 Y 座標 (4 の倍数を指定。0~1020)

XL : X の桁数-1 (8 ドット単位。0~127)

YL : Y の桁数-1 (16 ドット単位。0~63)

ただし、XS・YS 両方が-1だった場合、あるいは XL・YL 両方が-1だった場合には、先頭座標、桁・行はそれぞれ直前までの値が引きつがれます。

**戻 り 値**

上位 16 ビットには直前の XL、下位 16 ビットには直前の YL が返されます。

# B\_CUROFF

レベル 0

**書式** # include <iocslib.h>

void B\_CUROFF ( );

**機能** カーソルを消します。

**戻り値** 戻り値はありません。

# B\_CURON

レベル 0

**書 式** # include <iocslib.h>

```
void B_CURON ( );
```

**機 能** カーソルを表示します。

**戻 り 値** 戻り値はありません。



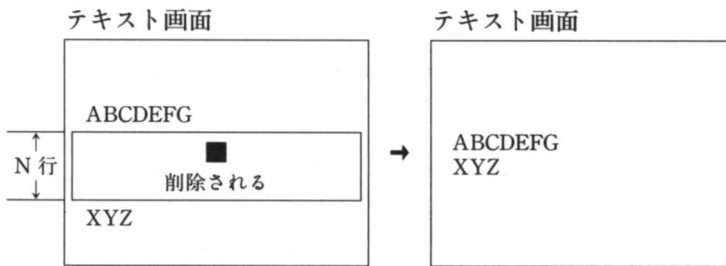
# B\_DEL

レベル 0

**書 式** # include <iocslib.h>

```
void B_DEL (N) ;  
int N; /* 削除する行数 */
```

**機 能** カーソル位置の行から N 行削除します。  
N に 0 を指定すると、1 とみなされます。  
なお、カーソルの X 座標は 0 となります。



**戻り値** 戻り値はありません。

# B\_DOWN

レベル 0

**書式** # include <iocslib.h>

```
void B_DOWN (N) ;  
int N; /* 移動する行数 */
```

**機能**

カーソル位置を、N 行下へ移動します。  
カーソル位置が最下行を超えてもスクロールアップは行われません。  
なお、N に 0 を指定すると、1 とみなされます。

**戻り値**

戻り値はありません。

# B\_DOWN\_S

レベル 0

**書 式** # include <iocslib. h>

void B\_DOWN\_S ( ) ;

**機 能** カーソル位置を、1 行下へ移動します。  
カーソル位置が最下行にあった場合には、スクロールアップが行われます。

**戻 り 値** 戻り値はありません。

# B\_DRVCHK

レベル 0

**書 式** # include <ioclib.h>

```
int B_DRVCHK (DRIVE, MODE) ;
int DRIVE; /* ドライブ番号 */
int MODE; /* モード */
```

**機 能** DRIVE で指定したドライブの状態のチェックと設定を行います (2HD のみ)。  
DRIVE には pda×256 を指定します。

- pda
  - 0x90~0x93 2HD フロッピーディスク

MODE には次の内容を設定します。

- 0 : 状態のチェックのみ行う (戻り値のビット 7~ビット 0)
- 1 : イジェクトする (イジェクト禁止状態のときは不可)
- 2 : イジェクト禁止 1 (MODE = 1 のイジェクトも禁止)
- 3 : イジェクト許可 1
- 4 : ディスクがセットされていないとき LED 点滅
- 5 : ディスクがセットされていないとき LED 消灯
- 6 : イジェクト禁止 2 (MODE = 1 のイジェクトも禁止)
- 7 : イジェクト許可 2
- 8 : 前回のチェック後にイジェクトしたか否かのチェック

上記の 6~8 は Human68K で使用するので、ユーザーは使用してはいけません。

**戻 り 値** MODE = 0~7 のときは、次の内容を返します。

ビット

7            6            5            4            3            2            1            0

LED 点滅	EJECT 禁止	BUFFER 有り	USER 使用禁止	PRO	READY	メディア 挿入	誤挿入

PRO (プロテクト = 1)、READY (ノットレディ = 1) は MODE = 0 で、  
メディア挿入のときにのみ返します。

MODE = 8 のとき、戻り値が 1 ならばイジェクトしていないことを、-1 ならばイジェクトしたことを示します。

# B\_DRVSN

レベル 0

**書式** # include <iocslib.h>

```
int B_DRVSN (DRIVE) ;
int DRIVE; /* ドライブ番号 */
```

**機能** ディスクのステータスを調べます。  
DRIVE には pda×256 を指定します。

● pda

- 0x80~0x8F    ハードディスク
- 0x90~0x93    2HD フロッピーディスク

**戻り値** 0xFFFFFFFF??ならばエラーを表します。  
正常終了ならばハードディスクは正の数 (0 も) を、2HD フロッピーディスクは最上位 8 ビットに FDC のステータス情報 (負の数もありえる) を返します。

戻り値

LED 点滅	EJECT BUFFER 禁止	USER 使用禁止	PRO READY	READY	マシ	人
7	6	5	4	3	2	1
0						

MODE = 8 のとき、戻り値が 1 ならば FDC のステータス情報が返ります。  
PRO (マシ) = 1, READY (マシ) = 1, は MODE = 0 のとき、マシの番号を返します。

# B\_DSKINI

レベル 0

&lt;2HD フロッピーの場合&gt;

書 式

```
# include <iocslib.h>

int B_DSKINI (DRIVE, DATAADDRESS, OFFTIME) ;
int DRIVE; /* ドライブ番号 */
unsigned char *DATAADDRESS; /* データアドレス */
int OFFTIME; /* モーターオフまでの時間 */
```

機 能

2HD フロッピーディスクのインターフェイスの初期化を行います。  
DRIVE には pda×256 を指定します。

- pda

0x90~0x93 2HD フロッピーディスク

DATAADDRESS には、SPECIFY コマンドのデータアドレスを指定します。

0 の場合はデフォルト値 (0x03, 0xd0, 0x10) となります。

OFFTIME にはモーターオフまでの時間 (n/100 秒) を指定します。

0 の場合はデフォルト値 (200 … 2 秒) となります。

戻 り 値

0xFFFFFFFF?? ならばエラーを表します。正常終了ならば正の数 (0 も) を返します。

&lt;ハードディスクの場合&gt;

書 式

```
# include <iocslib.h>

int B_DSKINI (DRIVE, DATAADDRESS) ;
int DRIVE; /* ドライブ番号 */
unsigned char *DATAADDRESS; /* データアドレス */
```

機 能

ハードディスクのインターフェイスの初期化を行います。

DRIVE には pda×256 を指定します。

- pda

0x80~0x8F ハードディスク

DATAADDRESS には、ドライブパラメータのデータアドレスを指定します。

0 の場合はデフォルト値 (0x01, 0x01, 0x00, 0x03, 0x01, 0x35, 0x80, 0x00, 0x00, 0x00) となります。

戻 り 値

0xFFFFFFFF?? ならばエラーを表します。正常終了ならば正の数 (0 も) を返します。

# B\_EJECT

レベル 0

**書式** # include <iocslib. h>

```
int B_EJECT (DRIVE) ;  
int DRIVE; /* ドライブ番号 */
```

**機能**

イジェクトを行います (イジェクト禁止状態でも実行します)。  
ハードディスクの場合は、未使用シリンダへヘッドをシークします。  
DRIVE には pda×256 を指定します。

● pda

0x80~0x8F ハードディスク  
0x90~0x93 2HD フロッピーディスク

**戻り値**

0xFFFFFFFF?? ならばエラーを表します。  
正常終了ならば正の数 (0 も) を返します。

# B\_ERA\_AL

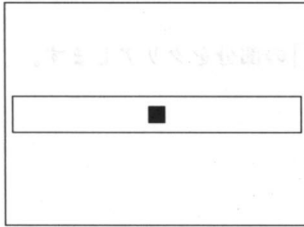
レベル 0

**書式** # include <iocslib.h>

void B\_ERA\_AL ( ) ;

**機能** テキスト画面のカーソルのある行の左端から右端までクリアします。

テキスト画面



の部分をクリアします。

**戻り値** 戻り値はありません。

# B\_ERASED

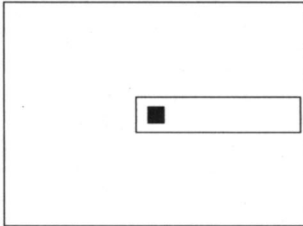
レベル 0

**書式** # include <iocslib.h>

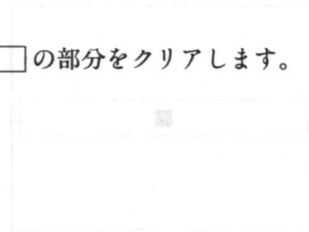
void B\_ERASED ( ) ;

**機能** テキスト画面のカーソル位置から、カーソルがある行の右端までをクリアします。

テキスト画面



の部分をクリアします。



**戻り値** 戻り値はありません。

# B\_ERA\_ST

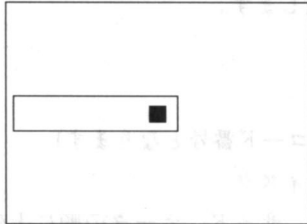
レベル 0

**書式** # include <ioclib.h>

```
void B_ERA_ST ( );
```

**機能** テキスト画面のカーソルがある行の左端から、カーソル位置までをクリアします。

テキスト画面



この部分をクリアします。

**戻り値** 戻り値はありません。

0	0	0	0	seek	retry	tim/mim	0
---	---	---	---	------	-------	---------	---

**留意**

# B\_FORMAT

レベル 0

**書 式**

```
# include <ioclib.h>

int B_FORMAT (DRIVE, RECNO, LENGTH, IDADDRESS) ;
int DRIVE; /* ドライブ番号 */
int RECNO; /* レコード番号 */
int LENGTH; /* バイト数 */
unsigned char * IDADDRESS; /* データアドレス */
```

**機 能**

ディスクの物理フォーマットを行います。  
DRIVE には  $pda \times 256 + mode$  を指定します。

● pda

0x80~0x8F ハードディスク

(RECNO は 256 バイト単位のレコード番号となります)

0x90~0x93 2HD フロッピーディスク

(RECNO はセクタ長、トラック、サイド、データの順に上位から 8 ビットずつで指定します)

● mode

mode にはハードディスクの場合、0 を指定します。

2HD フロッピーディスクの場合は、以下のようになります。

0	fm/mfm	retry	seek	0	0	0	0
---	--------	-------	------	---	---	---	---

LENGTH にはバイト数を指定します。

ただしハードディスクの場合は、インタリーブコード (6, 1) を指定します (通常 10 メガバイトドライブは 6、20・40 メガバイトドライブは 1)。

IDADDRESS には、ID データの先頭アドレスを指定します。

ただし、ハードディスクの場合は不要です。

IDADDRESS にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

**戻 り 値**

0xFFFFFFFF?? ならばエラーを表します。正常終了ならばハードディスクは正の数 (0 も) を、2HD フロッピーディスクは FDC のステータス情報 (負の数もありえる) を返します。

また、2HD フロッピーディスクの場合、物理フォーマットに失敗してもエラーにはならず、FDC のステータス情報として失敗したことを示す情報が返されます。

# BGCTRLGT

レベル0

**書 式** #include <iocslib.h>

```
int BGCTRLGT (MODE)
int MODE; /* モード */
```

**機 能** BG コントロールレジスタの読み出しを行います。  
MODE には BG コントロールレジスタ番号を指定します (0~1)。

**戻り値** テキストページ×2+表示モード (0~1)、またはエラーコード (-1=画面モードエラー、画面サイズが768×512のときなど) を返します。

# BGCTRLST

レベル 0

**書 式** #include <iocslib.h>

```
int BGCTRLST (MODE, PAGE, ONOFF) ;  
int MODE; /* モード */  
int PAGE; /* テキストページ */  
int ONOFF; /* 表示の ON/OFF */
```

**機 能** BG コントロールレジスタの設定を行います。

MODE には BG コントロールレジスタ番号を指定します (0~1)。

PAGE には、テキストページを指定します (0~1)。

-1 が指定された場合には、設定を変更せずに現在値を使用します。

ONOFF には表示の ON/OFF を指定します (0 のとき表示 OFF、1 のとき表示 ON)。

-1 が指定された場合には、設定を変更せずに現在値を使用します。

**戻 り 値** 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)

# BGSCRLGT

レベル 0

**書 式** #include <iocslib.h>

```
int BGSCRLGT (MODE, X, Y);
int MODE; /* モード */
int *X; /* X 座標を指すポインタ */
int *Y; /* Y 座標を指すポインタ */
```

**機 能**

BG スクロールレジスタの読み出しを行います。  
MODE には、BG スクロールレジスタ番号を指定します (0~1)。  
X には X 座標、Y には Y 座標を返すべきポインタをそれぞれ指定します。

**戻 り 値**

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)

正常終了の場合、X 座標、Y 座標がそれぞれのポインタに格納されます。

# BGSCRLST

レベル 0

**書 式** #include <iocslib.h>

```
int BGSCRLST (MODE, X, Y);  
int MODE; /* 垂直帰線期間の検出モード */  
int X; /* X 座標 */  
int Y; /* Y 座標 */
```

**機 能** BG スクロールレジスタの設定を行います。

MODE は次に示すように指定します。

ビット 31 : 0 垂直帰線期間を検出してから設定を行う

          : 1 垂直帰線期間を検出しないで設定を行う

ビット 0 : 0 BG0 を設定する

          : 1 BG1 を設定する

X には X 座標、Y には Y 座標をそれぞれ 0~1023 の範囲内で指定します。

-1 が指定された場合には、設定を変更せずに現在値を使用します。

**戻 り 値** 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)

# BGTEXTCL

レベル 0

**書 式** #include <iocslib. h>

```
int BGTEXTCL (PAGE, CODE);
int PAGE; /* テキストページ */
int CODE; /* パターンコード */
```

**機 能** BG テキストを指定したパターンでクリアします。

PAGE には、テキストページを指定します (0~1)。

CODE には、パターンコードを指定します。

このパターンコードとは、次の V、H、COLOR、PCGCODE の 4 つのパラメータを各ビットに割り当てたパターンのことです。

ビット 15 V : 0 縦方向反転しない

V : 1 縦方向反転する

ビット 14 H : 0 横方向反転しない

H : 1 横方向反転する

ビット 8~11 COLOR パレットブロック (0~15)

ビット 0~7 PCGCODE PCG コード (0~255)

※ビット 12、13、16~31 は常に 0

したがって、CODE=0x873C の時は、PCG コードが 60、パレットブロックが 7、縦方向のみ反転、という設定になります。

**戻り値** 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)

# BGTEXTGT

レベル 0

**書 式** #include <iocslib. h>

```
int BGTEXTGT (PAGE, X, Y) ;
int PAGE; /* テキストページ */
int X; /* テキスト X 座標 */
int Y; /* テキスト Y 座標 */
```

**機 能**

BG テキストの読み出しを行います。  
PAGE には、テキストページを指定します (0~1)。  
X にはテキスト X 座標、Y にはテキスト Y 座標を、それぞれ 0~63 の範囲内で指定します。

**戻 り 値**

正常終了の時はパターンコードを返します。  
パターンコードとは、次の 4 つのデータを各ビットに割り当てたパターンのことです。

ビット 15 V : 0 縦方向反転していない  
V : 1 縦方向反転している  
ビット 14 H : 0 横方向反転していない  
H : 1 横方向反転している  
ビット 8~11 COLOR パレットブロック (0~15)  
ビット 0~7 PCGCODE PCG コード (0~255)  
※ビット 12、13、16~31 は常に 0

異常終了の時はエラーコード (-1) を返します。  
これは画面モードが不正 (画面サイズが 768×512 の時など) な場合です。

# BGTEXTST

レベル 0

**書式** #include <ioclib.h>

```
int BGTEXTST (PAGE, X, Y, CODE) ;
int PAGE; /* テキストページ */
int X; /* テキスト X 座標 */
int Y; /* テキスト Y 座標 */
int CODE; /* パターンコード */
```

**機能**

BG テキストの設定を行います。

PAGE にはテキストページを指定します (0~1)。

X にはテキスト X 座標、Y にはテキスト Y 座標をそれぞれ 0~63 の範囲内で指定します。

CODE にはパターンコードを指定します。

このパターンコードとは、次の V, H, COLOR, PCGCODE の 4 つのパラメータを各ビットに割り当てたパターンのことです。

ビット 15 V : 0 縦方向反転しない

V : 1 縦方向反転する

ビット 14 H : 0 横方向反転しない

H : 1 横方向反転する

ビット 8~11 COLOR パレットブロック (0~15)

ビット 0~7 PCGCODE PCG コード (0~255)

※ビット 12、13、16~31 は常に 0

**戻り値**

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)

# BINDATEBCD

レベル 0

**書 式** #include <iocslib.h>

```
int BINDATEBCD (BINDATE) ;
int BINDATE; /* 日付 */
```

**機 能** 2進数の日付を、時計にセットできる形式に変換します。  
BINDATE の形式と、設定内容は次の通りです。

```
0000 yyyy yyyyyyyy mmmmmmmm dddddddd
```

yyyyyyyyyyy : 年 (1980~2079)

mmmmmmm : 月 (01~12)

ddddddd : 日 (01~31)

**戻り値** 次の内容を返します。

```
UUUUWWWW YYYYYYYY MMMMMMMM DDDDDDDD
```

UUUU : うるう年カウンタ (0~3)

WWWW : 曜日カウンタ (0~6) 0:日曜日

YYYYYYY : 年 (00~99。1980年から相対年数)

MMMMMMM : 月 (01~12)

DDDDDDD : 日 (01~31)

上記の年、月、日はいずれも BCD2桁です。  
エラーが発生したときには-1を返します。

## プログラム例

```

#include      <iocslib.h>

#define YEAR      1990
#define MONTH     3
#define DATE      14

main()
{
    int    bin;          /* 日付(2進数) */
    int    bcd;         /* 日付(BCD) */

    bin = (YEAR << 16) | (MONTH << 8) | (DATE << 0);

    bcd = BINDATEBCD(bin);

    BINDATESET(bcd);    /* 日付を時計に設定 */
}

```

# BINDATEGET

レベル 0

**書式** #include <iocslib.h>

```
int BINDATEGET ( );
```

**機能** 時計から日付を読み込みます。

**戻り値** 次の内容を返します。

```
0000 WWWY YYYYYYYY MMMMMMMM DDDDDDDD
```

WWWY : 曜日カウンタ (0~6) 0:日曜日

YYYYYYY : 年 (00~99. 1980年からの相対年数)

MMMMMMM : 月 (01~12)

DDDDDDD : 日 (01~31)

上記の年, 月, 日はいずれも BCD2 桁です。

## プログラム例

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

main()
{
    int bcd; /* 日付(BCD) */
    int bin; /* 日付(2進数) */

    bcd = BINDATEGET();
    bin = DATEBIN(bcd);

    printf("day = %d\n", (bin >> 28) & 0x0f);
    printf("year = %d\n", (bin >> 16) & 0x0fff);
    printf("month = %d\n", (bin >> 8) & 0xff);
    printf("date = %d\n", (bin >> 0) & 0xff);
}
```

# BINDATESET

レベル 0

**書式** #include <iocslib.h>

```
void BINDATESET (BCDDATE) ;
int BCDDATE; /* 日付 */
```

**機能** 時計に日付を設定します。

BCDDATE の形式と指定内容は次の通りです。

UUUUWWWW YYYYYYYY MMMMMMMM DDDDDDDD

UUUU : うるう年カウンタ (0~3)

WWWW : 曜日カウンタ (0~6) 0 : 日曜日

YYYYYYYY : 年 (00~99. 1980 年からの相対年数)

MMMMMMMM : 月 (01~12)

DDDDDDDD : 日 (01~31)

上記の年, 月, 日はいずれも BCD2 桁です。

**戻り値** 戻り値はありません。**プログラム例**

```
#include <iocslib.h>

#define YEAR 1990
#define MONTH 3
#define DATE 14

main()
{
    int bin; /* 日付(2進数) */
    int bcd; /* 日付(BCD) */

    bin = (YEAR << 16) | (MONTH << 8) | (DATE << 0);
    bcd = BINDATEBCD(bin);
    BINDATESET(bcd); /* 日付を時計に設定 */
}
```

# BINDNO

レベル 1

**書 式** # include <doslib.h>

```
int BINDNO (file1, file2) ;
unsigned char *file1; /* 実ファイル名へのポインタ */
unsigned char *file2; /* モジュールファイル名へのポインタ */
```

**機 能**

ファイル名からモジュール番号を求めます。  
file2 は、file1 で指定されたファイルに含まれる、ファイルのファイル名へのポインタです。  
オーバーレイ X ファイルに含まれる個々のファイルを実行するには、各ファイルに割り当てられたモジュール番号を指定しなければなりません。  
その時は、この BINDNO 関数でモジュール番号を求めることができます。

**戻 り 値**

file2 で指定されたファイルのモジュール番号をビット 15~8 に返します。  
戻り値が負の場合はエラーコードを示します。

**プログラム例**

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

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

    if (argc < 3) {
        printf("Usage: bindno file module1%n");
        exit(1);
    }

    file = *++argv;
    --argc;
    while (--argc) {
        module = *++argv;
        if ((mno = BINDNO(file, module)) >= 0) {
            printf("%s: %s mno = %d%n",
                file, module, mno);
        }
    }
    exit(0);
}
```

# B\_INS

レベル0

**書式** #include <iocslib.h>

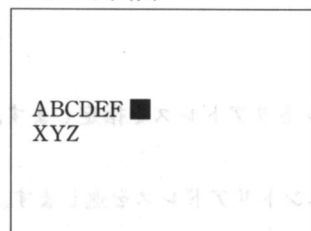
```
void B_INS (N) ;
int N; /* 挿入する行数 */
```

**機能** カーソル表示行の直後に N 行挿入します。

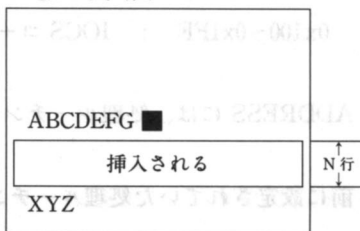
N に 0 を指定すると、1 とみなされます。

なお、カーソルの X 座標は 0 となります。

テキスト画面



テキスト画面



**戻り値** 戻り値はありません。

# B\_INTVCS

レベル 0

**書 式** # include <iocslib. h>

```
int B_INTVCS (VECTOR, ADDRESS) ;  
int VECTOR; /* ベクタ番号 */  
int ADDRESS; /* 処理ルーチンのエントリアドレス */
```

**機 能**

ベクタの設定を行います。  
VECTOR にはベクタ番号を指定します。

0~0xFF : 割り込み  
0x100~0x1FF : IOCS コール

ADDRESS には、処理ルーチンのエントリアドレスを指定します。

**戻 り 値**

前に設定されていた処理ルーチンのエントリアドレスを返します。

## BITSNS

## レベル 0

**書 式** #include <iocslib.h>

```
int BITSNS (KEYGRP) ;
int KEYGRP; /* キーコードグループ */
```

**機 能** 以下に示すキーコードグループを KEYGRP に指定して、キーの押下状態を調べます。

グループ	ビット							
	0	1	2	3	4	5	6	7
0	未定義	ESC	1 !	2 "	3 #	4 \$	5 %	6 &
1	7 '	8 (	9 )	0	- =	^	¥	BS
2	TAB	Q	W	E	R	T	Y	U
3	I	O	P	@	[	CR	A	S
4	D	F	G	H	J	K	L ;	
5	:	]	Z	X	C	V	B	N
6	M	, <	. >	/ ?	_	SP	HOME	DEL
7	Roll up	Roll down	UNDO	←	↑	→	↓	CLR
8	/	*	-	7	8	9	+	4
9	5	6	=	1	2	3	ENTER	0
A	,	.	記号	登録	HELP	XF1	XF2	XF3
B	XF4	XF5	かな	ローマ字	コード	CAPS	INS	ひらがな
C	全角	BREAK	COPY	F・1	F・2	F・3	F・4	F・5
D	F・6	F・7	F・8	F・9	F・10	未定義	未定義	未定義
E	SHIFT	CTRL	OPT.1	OPT.2	未定義	未定義	未定義	未定義
F	未定義	未定義	未定義	未定義	未定義	未定義	未定義	未定義

**戻り値** 各キーコードグループの押下状態を返します。  
該当ビットが1ならば、押下されていることを示します。

# B\_KEYINP

レベル 0

**書式** #include <iocslib.h>

int B\_KEYINP ( );

**機能** キーコードの読み出しを行います。

また、電卓が処理されます。

**戻り値** スキャンコード×256+内部コードを返します。

スキャンコードは、BITSNS 関数で指定するキーコードグループを8倍したものに、そのグループのキーのビット位置の番号を加えたものです。

例えば、'A'が押されていたならば、グループは3、ビット位置は6なので、 $3 \times 8 + 6 = 0x1E$  になります。

## プログラム例

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

main()
{
    int key_code; /* キーコード */

    key_code = B_KEYINP();
    printf("key_code = %d\n", key_code);
    printf("scan code = 0x%02x\n", key_code / 256);
    printf("internal code = 0x%02x\n", key_code % 256);
    printf("character = %c\n", key_code % 256);
}
```

# B\_KEYSNS

レベル 0

**書 式** # include <ioclib.h>

```
int B_KEYSNS ( ) ;
```

**機 能** キーの先行入力があるかないかを調べます。  
また、電卓が処理されます。

**戻 り 値** 0x010000+スキャンコード×256+内部コードを返します。  
0の場合には、キーが押下されていないことを示します。  
スキャンコードは、B\_KEYINP関数を参照してください。

# B\_LEFT

レベル 0

**書 式** # include <iocslib. h>

```
void B_LEFT (N)
int N; /* 移動する桁数 */
```

**機 能** カーソル位置を N 桁左へ移動します。  
カーソル位置が左端を超えても、右スクロールは行われません。  
なお、N に 0 を指定すると、1 とみなされます。

**戻り値** 戻り値はありません。

# B\_LOCATE

レベル 0

**書 式** # include <iocslib.h>

```
int B_LOCATE (X, Y) ;
int X; /* X座標 */
int Y; /* Y座標 */
```

**機 能** X と Y で指定した位置にカーソルを設定します。  
X に -1 を指定した場合には、前の座標の通知のみ行います。**戻 り 値** 上位 16 ビットに前の X 座標、下位の 16 ビットに前の Y 座標を返します。

# B\_LPEEK

レベル 0

**書 式** # include <iocslib.h>

```
int B_LPEEK (ADDRESS) ;  
unsigned long *ADDRESS; /* データの格納領域のアドレス */
```

**機 能** ADDRESS で指定されたアドレスから、データをロングワード単位 (4 バイト) で読み込みます。  
このときアドレスは偶数でなければなりません。

**戻 り 値** 読み込んだデータを返します。

## プログラム例

```
#include <stdio.h>  
#include <iocslib.h>  
  
static unsigned long mem = 0x100;  
  
main()  
{  
    unsigned long *addr; /* アドレス */  
    int rdata; /* 読み込んだデータ */  
  
    addr = &mem;  
  
    rdata = B_LPEEK(addr);  
    printf("%x: 0x%04x\n", addr, rdata);  
}
```



# B\_MEMSET

レベル 0

**書式** # include <iocslib.h>

```
void B_MEMSET (ADDRESS, DATABUF, DATABYTE);  
/* unsigned char *ADDRESS; /* データ書き込み領域のアドレス */  
/* unsigned char *DATABUF; /* データ格納領域のアドレス */  
/* int DATABYTE; /* データ書き込みバイト数 */
```

**機能** DATABUF が指す領域のデータを、ADDRESS で指定されたアドレスへ書き込みます。  
DATABYTE には、書き込むデータのバイト数-1 を指定します。  
このとき 2 つの領域を重ねてはいけません。

**戻り値** 戻り値はありません。

## プログラム例

```
#include <stdio.h>  
#include <iocslib.h>  
  
#define DATA_SIZE 100  
  
main()  
{  
    unsigned char src[DATA_SIZE]; /* 元データ */  
    unsigned char des[DATA_SIZE]; /* 転送先 */  
    int i;  
  
    for (i = 0; i < DATA_SIZE; i++)  
        src[i] = i;  
  
    B_MEMSET(des, src, DATA_SIZE - 1);  
  
    for (i = 0; i < DATA_SIZE; i++) {  
        if (i % 10 == 9)  
            printf("%02x\n", des[i]);  
        else  
            printf("%02x ", des[i]);  
    }  
}
```

# B\_MEMSTR

レベル 0

**書 式** # include <iocslib.h>

```
void B_MEMSTR (ADDRESS, DATABUF, DATABYTE) ;
unsigned char *ADDRESS; /* データ読み込み領域のアドレス */
unsigned char *DATABUF; /* データ格納領域のアドレス */
int DATABYTE; /* データ読み込みバイト数 */
```

**機 能** ADDRESS で指定されたアドレスからデータを読み込み、DATABUF が指す領域へ格納します。

DATABYTE には、読み込むデータのバイト数-1 を指定します。

このとき2つの領域を重ねてはいけません。

**戻 り 値** 戻り値はありません。

**プログラム例**

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

#define DATA_SIZE 100

main()
{
    unsigned char src[DATA_SIZE]; /* 元データ */
    unsigned char des[DATA_SIZE]; /* 転送先 */
    int i;

    for (i = 0; i < DATA_SIZE; i++)
        src[i] = i;

    B_MEMSTR(src, des, DATA_SIZE - 1);

    for (i = 0; i < DATA_SIZE; i++) {
        if (i % 10 == 9)
            printf("%02x\n", des[i]);
        else
            printf("%02x ", des[i]);
    }
}
```

# BOOTINF

レベル 0

**書 式** # include <ioclib.h>  
int BOOTINF( );

**機 能** パワー ON 情報とシステムのブート情報を返します。

**戻 り 値** 以下に示すブート情報を返します。

ビット 31~24: パワー ON 情報

0x00: パワースイッチにより起動

0x01: 外部スイッチにより起動

0x02: タイマにより起動

ビット 23~0: システムのブート情報

0x000080~0x00008F: ハードディスクより起動 (下位 4 ビットがブートしたドライブの番号を表す)

0x000090~0x000093: 2HD ディスクより起動 (下位 2 ビットがブートしたドライブの番号を表す)

0xED0000~0xED3FFF: SRAM からブートしたときのブートアドレス  
上記以外の値: ROM からブートしたときのブートアドレス

## プログラム例

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

main()
{
    long    inf;
    int     pow;
    long    bot;

    inf = BOOTINF();

    pow = (inf >> 24) & 0xff;
    switch (pow) {
    case 0x00:
        printf("パワースイッチにより起動\n");
        break;
    case 0x01:
        printf("外部スイッチにより起動\n");
        break;
    case 0x02:
        printf("タイマにより起動\n");
        break;
    }

    bot = inf & 0x00ffffff;
    if (0x000080 <= bot && bot <= 0x00008f)
        printf("boot from ハードディスク\n");
    else if (0x000090 <= bot && bot <= 0x000093)
        printf("boot from floppyディスク\n");
    else if (0xed0000 <= bot && bot <= 0xed3fff)
        printf("boot from SRAM\n");
    else
        printf("boot from ROM\n");
}
```

# BOX

## レベル 0

**書 式** #include <iocslib.h>

```
int BOX (ptr) ;
struct BOXPTR {
    short x1 ; /* 始点の X 座標 */
    short y1 ; /* 始点の Y 座標 */
    short x2 ; /* 終点の X 座標 */
    short y2 ; /* 終点の Y 座標 */
    unsigned short color ; /* パレットコード */
    unsigned short linestyle ; /* ラインスタイル */
} *ptr ;
```

**機 能** グラフィック画面にボックスを描画します。

**戻り値** 終了コードを返します。

```
0 : 正常終了
-1 : グラフィックは使用不可
```

# B\_PRINT

レベル 0

**書 式** # include <iocslib.h>

```
int B_PRINT (MSGPTR) ;  
unsigned char *MSGPTR; /* 文字列データを指すポインタ */
```

**機 能** ヌル文字で終了する文字列データを表示します。

**戻 り 値** 上位 16 ビットに表示後のカーソル X 座標、下位 16 ビットにカーソル Y 座標を返します。

# B\_PUTC

レベル 0

**書 式** # include <iocslib.h>

```
int B_PUTC (CODE) ;  
int CODE; /* 表示するデータ */
```

**機 能** CODE で指定した 1 バイトデータを表示します。

**戻 り 値** 上位 16 ビットに表示後のカーソル X 座標、下位 16 ビットにカーソル Y 座標を返します。

# B\_PUTMES

レベル 0

**書 式** # include <iocslib.h>

```
int B_PUTMES (COLOR, X, Y, MAX, ADDRESS) ;
int COLOR; /* 属性 */
int X; /* X座標 */
int Y; /* Y座標 */
int MAX; /* 最大表示文字数-1 */
unsigned char *ADDRESS; /* 表示文字列を指すポインタ */
```

**機 能**

指定位置（絶対座標）に、指定された文字列を表示します。

指定文字数を超えて表示されることはありません。

表示しようとした文字列が、全角文字の前半で切られてしまうような場合は、その全角文字は表示されず、空白が表示されます。

指定した文字数分表示する前にヌル文字が現れたら、以降の文字は空白で埋められます。

この関数は、ファンクション行表示用であり、B\_CONSOL には影響されません。また、表示後のカーソル位置にも変化はありません。

COLOR には属性を指定します。

X には X 座標（絶対座標）、Y には Y 座標（絶対座標）をそれぞれ指定します。

また、MAX には文字数-1（半角単位）を、ADDRESS には、ヌル文字で終了する文字列の先頭アドレスを指定します。

**戻り値**

次の X 座標の値を返します。

# B\_READ

レベル 0

```
書 式 # include <ioclib.h>

int B_READ (DRIVE, RECNO, LENGTH, ADDRESS) ;
int DRIVE; /* ドライブ番号 */
int RECNO; /* レコード番号 */
int LENGTH; /* バイト数 */
unsigned char * ADDRESS; /* データ読み込み領域の先頭アドレス */
```

機 能 ディスクからデータを読み込みます。  
DRIVE には  $pda \times 256 + mode$  を指定します。

- pda  
0x80~0x8F : ハードディスク  
(RECNO は 256 バイト単位のレコード番号となります)  
0x90~0x93 : 2HD フロッピーディスク  
(RECNO はセクタ長、トラック、サイド、セクタの順に上位から 8 ビットずつ設定します)
- mode  
mode にはハードディスクの場合、0 を指定します。  
2HD フロッピーディスクの場合は以下のようになります。

0	fm/mfm	retry	seek	0	0	0	0
---	--------	-------	------	---	---	---	---

LENGTH には、読み込みを行うデータのバイト数を指定します。

ハードディスクの場合は 256 の倍数を、2HD フロッピーディスクの場合は、1024 の倍数を指定してください。

ADDRESS には、データ読み込み領域の先頭アドレスを指定します。

ADDRESS にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

戻り値 0xFFFFFFFF?? ならばエラーを表します。正常終了ならばハードディスクは正の数 (0 も) を、2HD フロッピーディスクは FDC のステータス情報 (負の数もありえる) を返します。

また、2HD フロッピーディスクの場合、読み込みに失敗してもエラーにはならず、FDC のステータス情報として失敗したことを示す情報が返されます。

# B\_READDI

レベル 0

**書式**

# include &lt;iocslib.h&gt;

int B\_READDI (DRIVE, RECNO, LENGTH, ADDRESS) ;

int DRIVE; /\* ドライブ番号 \*/

int RECNO; /\* レコード番号 \*/

int LENGTH; /\* バイト数 \*/

unsigned char \*ADDRESS; /\* データ読み込み領域の先頭アドレス \*/

**機能**

診断のための読み出しを行います (2HDのみ)。

DRIVE には pda×256+mode を指定します。

## ● pda

0x90~0x93 : 2HD フロッピーディスク

(RECNO はセクタ長、トラック、サイド、セクタの順に上位から 8 ビットずつ指定します)

## ● mode

0	fm/mfm	retry	seek	0	0	0	0
---	--------	-------	------	---	---	---	---

LENGTH には読み出しを行うデータのバイト数を指定します。

1024 の倍数を指定してください。

ADDRESS には、データ読み込み領域の先頭アドレスを指定します。

ADDRESS には、スーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

**戻り値**

0xFFFFFFFF?? ならばエラーを表します。

正常終了ならば FDC のステータス情報 (負の数もありえる) を返します。

また、読み込みに失敗してもエラーにはならず、FDC のステータス情報として失敗したことを示す情報が返されます。

なお、CRC エラーなどが発生しても途中で中断して終了したりせずに、必ず指定バイト数の読み込みを行います。

# B\_READDL

レベル 0

```
# include <ioclib.h>

int B_READDL (DRIVE, RECNO, LENGTH, ADDRESS) ;
int DRIVE; /* ドライブ番号 */
int RECNO; /* レコード番号 */
int LENGTH; /* バイト数 */
unsigned char * ADDRESS; /* データ読み込み領域の先頭アドレス */
```

**機能** 削除データを読み込みます (2HD のみ)。  
DRIVE には  $pda \times 256 + mode$  を指定します。

- pda  
0x90~0x93 : 2HD フロッピーディスク  
(RECNO はセクタ長、トラック、サイド、セクタの順に上位から 8 ビットずつ設定します)
- mode

0	fm/mfm	retry	seek	0	0	0	0
---	--------	-------	------	---	---	---	---

LENGTH にはデータのバイト数を指定します。  
1024 の倍数を指定してください。  
ADDRESS には、データ読み込み領域の先頭アドレスを指定します。  
ADDRESS には、スーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

**戻り値** 0xFFFFFFFF?? ならばエラーを表します。正常終了ならば FDC のステータス情報 (負の数もありえる) を返します。  
また、読み込みに失敗してもエラーにはならず、FDC のステータス情報として失敗したことを示す情報が返されます。

# BREAKCK

## レベル 1

**書式** #include <doslib.h>

```
int BREAKCK (flag);
int flag; /* ブレークチェックの設定フラグ */
```

**機能** ブレークチェックの設定を行います。  
flag に指定できる内容は次の通りです。

- 0 : 特定のファンクションコールでのみチェックする
- 1 : すべてのファンクションコールでチェックする
- 1 : 設定状況の通知要求

**戻り値** 設定状況を表す次の値を返します。

- 0 : 特定のファンクションコールでのみチェックする
- 1 : すべてのファンクションコールでチェックする

### プログラム例

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

#define CHECK1 0 /* 指定のものをcheckするように設定 */
#define ALL_CK 1 /* すべてcheckするように設定 */
#define STATUS (-1) /* 設定状況のセンス */

main()
{
    if (BREAKCK(STATUS) == ALL_CK)
        puts("すべてのファンクションコールをチェック");
    else
        puts("指定のファンクションコールのみチェック");
}
```

# B\_RECALI

レベル 0

**書 式** # include <ioclib.h>

```
int B_RECALI (DRIVE) ;
int DRIVE; /* ドライブ番号 */
```

**機 能**

トラック 0 ヘシークを行います。  
DRIVE には pda×256 を指定します。  
pda×256+255 のときは、強制レディ状態での調査を行います (2HD のときのみ)。

● pda

0x80~0x8F : ハードディスク

0x90~0x93 : 2HD フロッピーディスク

**戻 り 値**

0xFFFFFFFF?? ならばエラーを表します。正常終了ならばハードディスクは正の数 (0 も) を、2HD フロッピーディスクは FDC のステータス情報 (負の数もありえる) を返します。

また、2HD フロッピーディスクの場合、シーク動作に失敗してもエラーにはならず、FDC のステータス情報として失敗したことを示す情報が返されます。

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

#define CHECK 0
#define ALL_OK 1
#define STATUS (-1)

main()
{
    if (BREAKER(STATUS) == ALL_OK)
        puts("すべてのドライブが正常です");
    else
        puts("一部のドライブが正常ではありません");
}
```



# B\_SEEK

レベル 0

**書式** # include <iocslib. h>

```
int B_SEEK (DRIVE, RECNO) ;
int DRIVE; /* ドライブ番号 */
int RECNO; /* レコード番号 */
```

**機能** 指定トラック (レコード) までシークします。

DRIVE には  $pda \times 256 + mode$  を指定します。

● pda

0x80~0x8F : ハードディスク

(RECNO は 256 バイト単位のレコード番号となります)

0x90~0x93 : 2HD フロッピーディスク

(RECNO はセクタ長、トラック、サイド、セクタの順に上位から 8 ビットずつ設定します)

● mode

mode にはハードディスクの場合、0 を指定します。

2HD フロッピーディスクの場合は以下のようになります。

0	fm/mfm	retry	seek	0	0	0	0
---	--------	-------	------	---	---	---	---

**戻り値**

0xFFFFFFFF?? ならばエラーを表します。正常終了ならばハードディスクは正の数 (0 も) を、2HD フロッピーディスクは FDC のステータス情報 (負の数もありえる) を返します。

また、2HD フロッピーディスクの場合、シーク動作に失敗してもエラーにはならず、FDC のステータス情報として失敗したことを示す情報が返されます。

# B\_SFTSNS

レベル 0

**書式** # include <iocslib. h>

```
int B_SFTSNS ( );
```

**機能** シフトキーの押下状態を調べます。

**戻り値** シフトキーの押下状態を返します。

押下状態を示す各ビットの内容は次の通りで、ビット = 1 ならば各シフトキーが押下されたことを示します。

●ビット 15~8

15      14      13      12      11      10      9      8

0	全角	ひらがな	INS	CAPS	コード入力	ローマ字	かな
---	----	------	-----	------	-------	------	----

●ビット 7~0

7      6      5      4      3      2      1      0

CAPS	コード入力	ローマ字	かな	OPT2	OPT1	CTRL	SHIFT
------	-------	------	----	------	------	------	-------

上記のうち、LED の付いているキーは、LED の点灯・消灯の状態を、LED の付いていないキーは実際に押下されているかどうかの状態を示します。

# B\_SUPER

レベル 0

**書 式** # include <iocslib.h>

```
int B_SUPER (SSP) ;  
int SSP ;
```

**機 能**

スーパーバイザモード、ユーザーモード間の切り替えを行います。

SSP に 0 を指定するとユーザーモードからスーパーバイザモードに替わります。

SSP に 0 以外を指定するとスーパーバイザモードからユーザーモードに替わります。

**戻 り 値**

前の SSP を返します。

すでにスーパーバイザモードとなっている状態で、再度スーパーバイザモードに切り替えを行うように指定するとエラーになり、負の数を返します。

# B\_UP

レベル 0

**書式** # include <iocslib.h>

```
void B_UP (N) ;  
int N ; /* 移動する行数 */
```

**機能** カーソル位置を N 行上へ移動します。  
カーソル位置が先頭行を超えてもスクロールダウンは行われません。  
なお、N に 0 を指定すると、1 とみなされます。

**戻り値** 戻り値はありません。

# B\_UP\_S

レベル 0

**書 式** # include <iocslib. h>

void B\_UP\_S ( ) ;

**機 能** カーソル位置を 1 行上へ移動します。

カーソル位置が先頭行にあった場合には、スクロールダウンが行われます。

**戻 り 値** 戻り値はありません。

# BUS\_ERR

## レベル 1

**書 式** # include <doslib. h>

```
int BUS_ERR(s_adr, d_adr, mode) ;  
unsigned char *s_adr; /* 読み込みポインタ */  
unsigned char *d_adr; /* 書き込みポインタ */  
int mode; /* アクセスする時の単位サイズ */
```

**機 能**

スーパーバイザ領域や、何もマッピングされていない領域など、アクセスした時にバスエラーが発生する可能性のある領域を、読み書きできるかどうかをチェックして、その結果を返します。

s\_adr には、読み込みたい領域へのポインタを指定します。

d\_adr には、書き込みたい領域へのポインタを指定します。

mode は、アクセスする時の単位サイズを次のように設定します。

- 1: バイトでアクセス
- 2: ワードでアクセス
- 4: ロングワードでアクセス

**戻 り 値**

戻り値が 0 なら、読み書き両方とも可能であることを示し、読み込みポインタで示されるアドレスからデータを読み込んで、書き込みポインタで示されるポインタへデータを書き込んだことを示します。

1 なら、d\_adr で指定したアドレスに書き込みを行ったときにバスエラーが発生することを示します。

2 なら、s\_adr で指定したアドレスで読み込みを行ったときにバスエラーが発生することを示します。

-1 なら、おかしい引数を指定したことを示します(mode が不正、あるいは奇数アドレスからワード、ロングワードでアクセスするよう指定したとき)。

s\_adr を先にチェックしますので、s\_adr, d\_adr 両方でバスエラーが発生する場合には 2 を返します。

## プログラム例

```

#include      <stdio.h>
#include      <doslib.h>

#define M_BYTE      1
#define M_WORD      2
#define M_LONG      4
#define R_OK        0
#define R_WRITE     1
#define R_READ      2
#define R_ERR       (-1)

main()
{
    char      *s_adr;
    char      *d_adr;
    int       mode;
    int       r;

    mode = M_BYTE;
    s_adr = (char *)0xc00000;
    d_adr = (char *)0xe00000;
    r = BUS_ERR(s_adr, d_adr, mode);
    switch (r) {
    case R_OK:
        printf("success\n");
        break;
    case R_WRITE:
        printf("can't write\n");
        break;
    case R_READ:
        printf("can't read\n");
        break;
    case R_ERR:
        printf("error\n");
        break;
    }
}

```

ルビ

# B\_VERIFY

レベル 0

**書 式** # include <ioclib.h>

```
int B_VERIFY (DRIVE, RECNO, LENGTH, ADDRESS) ;
/*
int DRIVE; /* ドライブ番号 */
int RECNO; /* レコード番号 */
int LENGTH; /* バイト数 */
unsigned char * ADDRESS; /* 比較するデータの先頭アドレス */
```

**機 能** データの比較チェックを行います。

DRIVE には  $pda \times 256 + mode$  を指定します。

● pda

0x80~0x8F : ハードディスク

(RECNO は 256 バイト単位のレコード番号となります)

0x90~0x93 : 2HD フロッピーディスク

(RECNO はセクタ長、トラック、サイド、セクタの順に上位から 8 ビットずつ設定します)

● mode

mode にはハードディスクの場合、0 を指定します。

2HD フロッピーディスクの場合は以下のようになります。

0	fm/mfm	retry	seek	0	0	0	0
---	--------	-------	------	---	---	---	---

LENGTH には比較チェックを行うデータのバイト数を指定します。ハードディスクの場合は 256 の倍数を、2HD フロッピーディスクの場合は 1024 の倍数を指定してください。

ADDRESS には、比較するデータの先頭アドレスを指定します。

ADDRESS には、スーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

**戻り値**

0xFFFFFFFF?? ならばエラーを表します。正常終了ならばハードディスクは正の数 (0 も) を、2HD フロッピーディスクは FDC のステータス情報 (負の数もありえる) を返します。

また、2HD フロッピーディスクの場合、比較チェックに失敗してもエラーにはならず、FDC のステータス情報として失敗したことを示す情報が返されます。

# B\_WPEEK

レベル 0

**書式** # include <iocslib.h>

```
int B_WPEEK (ADDRESS) ;  
unsigned short *ADDRESS ; /* データ格納領域のアドレス */
```

**機能** ADDRESS で指定されたアドレスから、データをワード単位 (2 バイト) で読み込みます。  
このときアドレスは偶数でなければなりません。

**戻り値** 読み込んだデータを返します。

## プログラム例

```
#include <stdio.h>  
#include <iocslib.h>  
  
static unsigned short mem = 0x2000;  
main()  
{  
    unsigned short *addr; /* アドレス */  
    int rdata; /* 読み込んだデータ */  
  
    addr = &mem;  
  
    rdata = B_WPEEK(addr);  
    printf("0x%x: 0x%02x%n", addr, rdata);  
}
```

# B\_WPOKE

レベル 0

**書 式** #include <iocslib.h>

```
void B_WPOKE (ADDRESS, DATA) ;
unsigned short *ADDRESS; /* データ格納領域のアドレス */
int DATA; /* 書き込むデータ */
```

**機 能** ADDRESS で指定されたアドレスへ、データをワード単位 (2 バイト) で書き込みます。

このときアドレスは偶数で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 <iocslib.h>

static unsigned short mem = 0x100;

main()
{
    unsigned short *addr; /* アドレス */
    int rdata; /* 読み込んだデータ */

    addr = &mem;

    rdata = B_WPEEK(addr);
    printf("0x%02x%02x\n", addr, rdata);

    B_WPOKE(addr, 0x2000);

    rdata = B_WPEEK(addr);
    printf("0x%02x%02x\n", addr, rdata);
}
```

# B\_WRITE

レベル 0

## 書式

```
#include <ioclib.h>
```

```
int B_WRITE (DRIVE, RECNO, LENGTH, ADDRESS) ;
```

```
int DRIVE; /* ドライブ番号 */
```

```
int RECNO; /* レコード番号 */
```

```
int LENGTH; /* バイト数 */
```

```
unsigned char * ADDRESS; /* 書き出すデータの先頭アドレス */
```

## 機能

ディスクにデータを書き出します。

DRIVE には  $pda \times 256 + mode$  を指定します。

### ● pda

0x80～0x8F : ハードディスク

(RECNO は 256 バイト単位のレコード番号となります)

0x90～0x93 : 2HD フロッピーディスク

(RECNO はセクタ長、トラック、サイド、セクタの順に上位から 8 ビットずつ設定します)

### ● mode

mode にはハードディスクの場合、0 を指定します。

2HD フロッピーディスクの場合は以下のようになります。

0	fm/mfm	retry	seek	0	0	0	0
---	--------	-------	------	---	---	---	---

LENGTH には書き出しを行うデータのバイト数を指定します。ハードディスクの場合は 256 の倍数を、2HD フロッピーディスクの場合は 1024 の倍数を指定してください。

ADDRESS には、書き出すデータの先頭アドレスを指定します。

ADDRESS には、スーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

## 戻り値

0xFFFFFFFF?? ならばエラーを表します。正常終了ならばハードディスクは正の数 (0 も) を、2HD フロッピーディスクは FDC のステータス情報 (負の数もありえる) を返します。

また、2HD フロッピーディスクの場合、書き出しに失敗してもエラーにはならず、FDC のステータス情報として失敗したことを示す情報が返されます。

# B\_WRITED

レベル 0

**書 式** #include <iocslib.h>

```
int B_WRITED (DRIVE, RECNO, LENGTH, ADDRESS);
int DRIVE; /* ドライブ番号 */
int RECNO; /* レコード番号 */
int LENGTH; /* バイト数 */
unsigned char *ADDRESS; /* 書き込みデータの先頭アドレス */
```

**機 能**

削除データを書き込みます (2HD のみ)。  
DRIVE には  $pda \times 256 + mode$  を指定します。

- pda

0x90~0x93 : 2HD フロッピーディスク

(RECNO はセクタ長、トラック、サイド、セクタの順に上位から 8 ビット  
ずつ設定します)

- mode

0	fm/mfm	retry	seek	0	0	0	0
---	--------	-------	------	---	---	---	---

LENGTH にはデータのバイト数を指定します。1024 の倍数を指定してください。  
ADDRESS には、書き込みデータの先頭アドレスを指定します。  
ADDRESS には、スーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

**戻り値**

0xFFFFFFFF?? ならばエラーを表します。正常終了ならば FDC のステータス情報  
(負の数もありえる) を返します。

また、書き込みに失敗してもエラーにはならず、FDC のステータス情報として失敗したことを示す情報が返されます。

# C\_CLS\_AL

レベル 1

**書式** #include <doslib.h>

```
int C_CLS_AL ( );
```

**機能**

テキスト画面全体をクリアします。  
 カーソルはホームポジションへ移動します。  
 このファンクションコールは、CON デバイスによりサポートされます。

**戻り値**

処理が正常に終了したときは、0 を返します。

**参照関数**

C\_CLS\_ED, C\_CLS\_ST

0	0	0	0	seek	tefy	tm\mfm	0
---	---	---	---	------	------	--------	---

戻り値

# C\_CLS\_ED

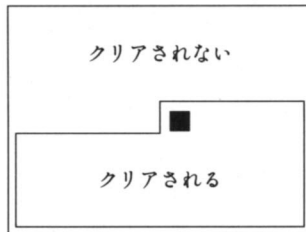
レベル 1

**書 式** #include <doslib.h>

```
int C_CLS_ED ( ) ;
```

**機 能** カーソル位置から最終行右端まで、テキスト画面をクリアします。  
このファンクションコールは、CON デバイスによりサポートされます。

テキスト画面



**戻り値** 処理が正常に終了したときは、0を返します。

**参照関数** C\_CLS\_ST, C\_CLS\_AL

# C\_CLS\_ST

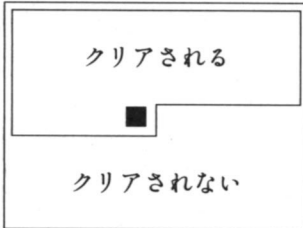
レベル 1

**書 式** #include <doslib.h>

```
int C_CLS_ST ( ) ;
```

**機 能** 先頭行左端からカーソル位置まで、テキスト画面をクリアします。  
このファンクションコールは、CON デバイスによりサポートされます。

テキスト画面



**戻 り 値** 処理が正常に終了したときは、0 を返します。

**参 照 関 数** C\_CLS\_ED, C\_CLS\_AL

# C\_COLOR

## レベル 1

**書 式** #include <doslib.h>

```
int C_COLOR (atr) ;
int atr; /* カラー属性 */
```

**機 能**

atr で指定した属性を設定します。  
カラー属性の内容は次の通りです。

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

このファンクションコールは、CON デバイスによりサポートされます。

**戻 り 値**

処理が正常に終了したときは、0 を返します。

# C\_CUROFF

レベル 1

**書 式** #include <doslib.h>

int C\_CUROFF ( ) ;

**機 能** カーソルを非表示モードにします。  
このファンクションコールは、CON デバイスによりサポートされます。

**戻 り 値** 処理が正常に終了したときは、0 を返します。

**参 照 関 数** C\_CURON

# C\_CURON

## レベル 1

**書式** #include <doslib.h>

```
int C_CURON ( );
```

**機能** カーソルを表示モードにします。

このファンクションコールは、CON デバイスによりサポートされます。

**戻り値** 処理が正常に終了したときは、0 を返します。

**参照関数** C\_CUROFF

# C\_DEL

レベル 1

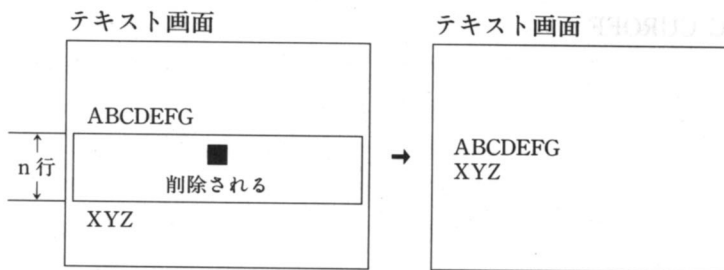
**書式** #include <doslib.h>

```
int C_DEL (n) ;  
int n; /* 削除する行数 */
```

**機能** カーソルのある行から n 行削除します。

n に 0 を指定した場合、1 とみなされます。

このファンクションコールは、CON デバイスによりサポートされます。



**戻り値** 処理が正常に終了したときは、0 を返します。

**参照関数** C\_INS

# C\_DOWN

## レベル 1

**書式** #include <doslib.h>

```
int C_DOWN (n) ;  
int n; /* カーソル位置の移動行数 */
```

**機能** カーソル位置を n 行下へ移動します。  
ただし、最終行より下へ移動しようとしても、スクロールアップは行われません。  
n に 0 を指定した場合、1 とみなされます。  
このファンクションコールは、CON デバイスによりサポートされます。

**戻り値** 処理が正常に終了したときは、0 を返します。

**参照関数** C\_UP

# C\_DOWN\_S

レベル 1

**書 式** #include <doslib.h>

```
int C_DOWN_S ( ) ;
```

**機 能** カーソル位置を 1 行下へ移動します。

最終行より下へ移動するときには、スクロールアップが行われます。

このファンクションコールは、CON デバイスによりサポートされます。

**戻 り 値** 処理が正常に終了したときは、0 を返します。

**参 照 関 数** C\_UP\_S

# C\_ERA\_AL

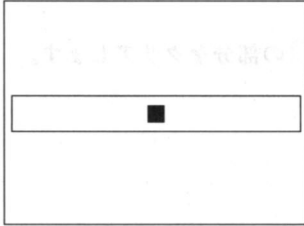
レベル 1

**書式** #include <doslib.h>

```
int C_ERA_AL ( );
```

**機能** テキスト画面のカーソルのある行の左端から、右端までをクリアします。  
このファンクションコールは、CON デバイスによりサポートされます。

テキスト画面



の部分をクリアします。

**戻り値** 処理が正常に終了したときは、0 を返します。

**参照関数** C\_ERA\_ED, C\_ERA\_ST

# C\_ERA\_ED

レベル 1

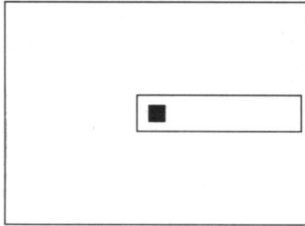
**書 式** #include <doslib.h>

```
int C_ERA_ED ( ) ;
```

**機 能** テキスト画面のカーソル位置から、カーソルがある行の右端までをクリアします。

このファンクションコールは、CON デバイスによりサポートされます。

テキスト画面



の部分をクリアします。

**戻り値** 処理が正常に終了したときは、0 を返します。

**参照関数** C\_ERA\_ST, C\_ERA\_AL

# C\_ERA\_ST

レベル 1

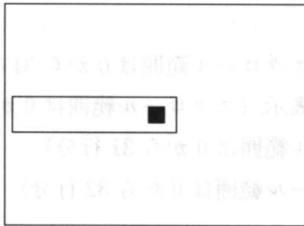
**書式** #include <doslib.h>

```
int C_ERA_ST ( );
```

**機能** テキスト画面のカーソルのある行の左端から、カーソル位置までをクリアします。

このファンクションコールは、CON デバイスによりサポートされます。

テキスト画面



の部分をクリアします。

**戻り値** 処理が正常に終了したときは、0 を返します。

**参照関数** C\_ERA\_ED, C\_ERA\_AL

# C\_FNKMOD

レベル 1

**書 式** #include <doslib.h>

```
int C_FNKMOD (n) ;  
int n; /* ファンクションキー行のモード */
```

**機 能** ファンクションキー行のモードを指定します。

スクロール範囲はリセットされます。

このファンクションコールは、CON デバイスによりサポートされます。

なお、n で指定できるモードは次の通りです。

- 0 : ファンクションキー表示 (スクロール範囲は 0 から 31 行分)
- 1 : シフトファンクションキー表示 (スクロール範囲は 0 から 31 行分)
- 2 : 何も表示しない (スクロール範囲は 0 から 31 行分)
- 3 : ふつうの行とする (スクロール範囲は 0 から 32 行分)
- 1 : 現在のモードを返す

**戻 り 値** n に -1 を指定したときのみ、現在のモード (0~3) を返します。

# CHANGE\_PR

レベル 1

**書 式** # include <doslib.h>

```
void CHANGE_PR( );
```

**機 能** バックグラウンドタスクの自分の実行権を放棄します。  
次のタスクに切り替わります。

**戻 り 値** 戻り値はありません。

**参 照 関 数** OPEN\_PR, KILL\_PR, GET\_PR, SUSPEND\_PR, SLEEP\_PR, SEND\_PR,  
TIME\_PR

# CHDIR

レベル 1

**書 式** #include <doslib.h>

```
int CHDIR (file) ;  
unsigned char *file; /* ディレクトリ名格納領域へのポインタ */
```

**機 能** カレントディレクトリを、fileで指定したディレクトリに変更します。

**戻 り 値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) 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は、0以外の正の数の場合があります。

**参 照 関 数** MKDIR, RMDIR

# CHGDRV

レベル 1

**書 式** #include <doslib.h>

```
int CHGDRV (drive) ;
```

```
/* int drive; /* ドライブ番号 */
```

**機 能** カレントドライブを指定します。

指定するドライブ番号とドライブの対応関係は次のようになります。

0 : ドライブ A

1 : ドライブ B

2 : ドライブ C

3 : ドライブ D

⋮

⋮

**戻り値** エラーならば指定した値以下の値を返します。

正常終了ならば指定した値より大きい値を返します。

## プログラム例

```
#include <doslib.h>

main()
{
    int drive; /* ドライブ番号 */

    drive = 1;
    printf("指定可能なドライブ数: %d\n", CHGDRV(drive));

    if (CURDRV() < drive)
        puts("ERROR");
}
```

# CHMOD

## レベル 1

**書 式** #include <doslib.h>

```
int CHMOD (file, atr) ;
unsigned char *file; /* ファイル名格納領域へのポインタ */
int atr; /* ファイル属性 */
```

**機 能**

file で指定するファイルの属性を変更します。  
 atr に指定できる属性は次の通りです。  
 ただし、atr に -1 を指定すると、属性の読み出しのみを行います。

- |       |            |   |
|-------|------------|---|
| ビット 0 | : 読み込み専用   | } 組み合わせで 0x00~0x07 まで指定できます。                  |
| 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 ID | } この3つのうちどれか1つを指定して上記ビット0~ビット2と組み合わせの値を指定します。 |
| ビット 4 | : ディレクトリ   |   |
| ビット 5 | : 通常ファイル   |   |

**戻り値**

負の数 (-1 以外も) でエラーとなります。  
 属性の変更に成功すると正の数 (0 も) を返します。  
 属性の読み出しに成功すると読み出した属性を返します。

```
<doslib.h>
#include
main()
{
    int drive;
    drive = 1;
    printf("指定可能なドライブ: %d", CHMOD(drive));
    if (CHMOD(drive) < drive)
        printf("ERROR");
}
```

# C\_INS

## レベル 1

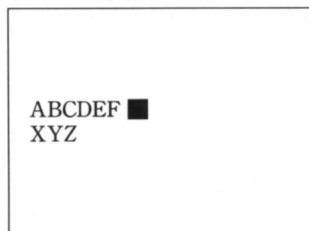
**書式** #include <doslib.h>

```
int C_INS (n) ;  
int n; /* 挿入する行数 */
```

**機能**

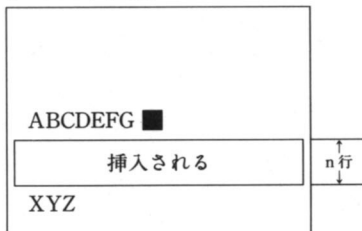
カーソルのある行の直前に n 行挿入します。  
n に 0 を指定した場合、1 とみなされます。  
このファンクションコールは、CON デバイスによりサポートされます。

テキスト画面



→

テキスト画面



**戻り値**

処理が正常に終了したときは、0 を返します。

**参照関数**

C\_DEL

# CINSNS

レベル 1

**書 式** #include <doslib.h>

```
int CINSNS ( ) ;
```

**機 能** RS-232C 回線からの入力可・不可を調べます。

**戻 り 値** 入力不可のときには0、入力可能ならば0以外を返します。

**参 照 関 数** COUTSNS



# CIRCLE

## レベル 1

**書 式** #include <iocslib.h>

```
int CIRCLE (circleptr) ;
struct CIRCLEPTR {
    short x; /* 中心 X 座標 */
    short y; /* 中心 Y 座標 */
    unsigned short radius; /* 半径 */
    unsigned short color; /* パレットコード */
    short start; /* 円弧開始角度 */
    short end; /* 円弧終了角度 */
    unsigned short ratio; /* 比率 */
} *circleptr;
```

**機 能**

グラフィック画面にサークルを描画します。

円弧開始角度、円弧終了角度は符号つき 2 バイト整数であり、0~360 の角度で円弧を描きます。

範囲外の角度を指定した場合は、360 を指定した事とみなされます。

負の値を指定すると扇型を描きます (角度は値の絶対値となります)。

比率は符号なし 2 バイト整数であり、0~65535 の値をとります。

比率が 0~255 のとき半径は横方向の半径を意味し、縦方向との比率は X:256 となります。

比率が 256~65535 のとき半径は縦方向の半径を意味し、横方向との比率は X:256 となります。

**戻 り 値**

終了コードを返します。

0 : 正常終了

-1 : グラフィックは使用不可

# C\_LEFT

# CIRCLE

レベル 1

**書 式** #include <doslib.h>

```
int C_LEFT (n) ;
int n; /* カーソル位置の移動桁数 */
```

**機 能** カーソル位置を n 桁左へ移動します。  
 ただし、最左端より左へ移動しようとしても、右スクロールは行われません。  
 n に 0 を指定した場合、1 とみなされます。  
 このファンクションコールは、CON デバイスによりサポートされます。

**戻 り 値** 処理が正常に終了したときは、0 を返します。

**参 照 関 数** C\_RIGHT

# CLIPPUT

レベル 0

**書 式** #include <iocslib. h>

```
void CLIPPUT (XDOT, YDOT, ADDRESS, CLIPPTR) ;
int XDOT; /* Xドット座標 */
int YDOT; /* Yドット座標 */
struct FNTBUF {
    short xl; /* 書き込むパターンの X 方向のドット数 */
    short yl; /* 書き込むパターンの Y 方向のドット数 */
    unsigned char buffer []; /* パターンデータ */
} *ADDRESS;
struct CLIPXY {
    short xs; /* 表示領域の先頭 X 座標 */
    short ys; /* 表示領域の先頭 Y 座標 */
    short xe; /* 表示領域の最終 X 座標+1 */
    short ye; /* 表示領域の最終 Y 座標+1 */
} *CLIPPTR;
```

**機 能**

ADDRESS で指定されたアドレスから、XDOT および YDOT で指定されたドット座標へのパターンを書き出すとともに、CLIPPTR が示すクリップ座標に従いクリッピング処理を行います。

ADDRESS にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないように注意してください。

ADDRESS には、書き出すパターンの X 方向のドット数、Y 方向のドット数、およびパターンデータを格納してください。

buffer には、指定した X、Y 方向のドット数に必要なだけのデータを格納しておいてください。

構造体 FNTBUF には、データバッファの宣言のみで実体は存在していません。メモリ確保用の関数 malloc など確保した領域を使用してください。

また CLIPPTR には、表示領域の先頭 X 座標、先頭 Y 座標、最終 X 座標+1、最終 Y 座標+1 を格納します。

**戻 り 値**

戻り値はありません。

# C\_LOCATE

レベル 1

**書式** #include <doslib.h>

```
int C_LOCATE (x, y);
int x;
int y;
```

**機能** カーソルを (x, y) で指定した位置に設定します。  
このファンクションコールは、CON デバイスによりサポートされます。

**戻り値** 処理が正常に終了したときは、0 を返します。

演 算

戻り値

# CLOSE

レベル 1

**書 式** #include <doslib. h>

```
int CLOSE (fileno) ;  
int fileno; /* ファイルハンドル */
```

**機 能** fileno で指定するファイルハンドルをクローズします。

**戻 り 値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) 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は、0 以外の正の数の場合があります。

# COMINP

レベル 1

**書 式** #include <doslib.h>

```
int COMINP ( ) ;
```

**機 能** RS-232C 回線から 1 バイト入力します。

また、このときブレークチェックも行います (^C)。

**戻 り 値** 入力データを返します。

**参 照 関 数** COMOUT, CINSNS, INP232C

# COMMON\_CK

## レベル 1

**書式** # include <doslib.h>

```
int COMMON_CK (name) ;
unsigned char *name; /* ブロックの名前 */
```

**機能** common 領域において、指定するブロックが存在するかどうかチェックします。  
name には、チェックしたいブロックの名前を指定します。

**戻り値** 指定したブロックが存在する場合は、そのブロックの大きさをバイト数で返します。  
存在しない場合は、エラーになり負の数を返します。

**参照関数** COMMON\_RD, COMMON\_WT, COMMON\_LK, COMMON\_FRE,  
COMMON\_DEL

### プログラム例

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

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

    if (argc == 2)
        name = argv[1];
    else {
        puts("COMMON 領域の名前を指定してください。");
        exit(1);
    }

    printf("COMMON 領域%s", name);
    if ((size = COMMON_CK(name)) < 0)
        printf("は存在しません。%n");
    else {
        printf("は存在します。%n");
        printf("領域の大きさは %dバイト%n", size);
    }
    exit(0);
}
```

# COMMON\_DEL

レベル 1

**書式** # include <doslib.h>

```
int COMMON_DEL (name) ;
unsigned char *name; /* ブロックの名前 */
```

**機能** common 領域において、指定されたブロックを消去します。  
name は消去したいブロックの名前を指定します。

**戻り値** 負の数ならエラーを表します。

**参照関数** COMMON\_CK, COMMON\_RD, COMMON\_WT, COMMON\_LK,  
COMMON\_FRE

## プログラム例

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

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

    if (argc == 2)
        name = argv[1];
    else {
        puts("COMMON 領域の名前を指定してください。");
        exit(1);
    }

    printf("COMMON 領域%s", name);
    if (COMMON_DEL(name) >= 0)
        printf("を削除しました。%n");
    else
        printf("の削除に失敗しました。%n");
    exit(0);
}
```

# COMMON\_FRE

レベル 1

**書 式** # include <doslib. h>

```
int COMMON_FRE (name, pos, id_psp, len) ;
unsigned char *name; /* ブロックの名前 */
int pos; /* ロック解除するデータの先頭を指す */
int id_psp; /* ロック解除するときの ID */
int len; /* ロック解除するデータの長さ */
```

**機 能**

common 領域において、指定されたブロック中のデータをロック解除します。ロック解除すると、id\_pspで指定されたプロセス以外からでもアクセスできるようになります。

name にはブロックの名前を指定します。

pos にはロックしたときに指定した位置を指定します。

id\_psp はロックしたときに付けたプロセス ID です。

また len はロックしたときに指定したバイト数です。

COMMON\_FRE で指定するパラメータは、COMMON\_LK で指定したパラメータと完全に同一でなければなりません。これは、ロックしたプロセス以外のプロセスがロック解除を不可能にするためです。

**戻 り 値**

負の数ならエラーを表します。

**参 照 関 数**

COMMON\_CK, COMMON\_RD, COMMON\_WT, COMMON\_LK,  
COMMON\_DEL

## COMMON\_FRE

## プログラム例

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

/* extern int _PSP; */

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

    if (argc == 2)
        name = argv[1];
    else {
        puts("COMMON領域の名前を指定してください。");
        exit(1);
    }

    printf("COMMON領域 %s", name);
    if ((size = COMMON_CK(name)) < 0)
        printf("は存在しません。%n");
    else if (COMMON_FRE(name, 0, _PSP, size) >= 0)
        printf("を%dでロック解除しました。%n", _PSP);
    else
        printf("のロック解除に失敗しました。%n");
    exit(0);
}
```

終了

戻る

# COMMON\_LK

レベル 1

**書 式** # include <doslib. h>

```
int COMMON_LK (name, pos, id_psp, len) ;
unsigned char *name; /* ブロックの名前 */
int pos; /* ロックするデータの先頭を指す */
int id_psp; /* ロックするときの ID */
int len; /* ロックするときのデータの長さ */
```

**機 能**

common 領域において、指定されたブロック中のデータをロックします。

ロックすると、id\_psp で指定されたプロセス以外からアクセスできなくなります。

name にはブロックの名前を指定します。

pos にはブロック中のデータのロックする位置を指定します。

id\_psp にはロックするプロセスのプロセス ID を指定します。

通常は自分自身の PSP を指定します。これはシステムで定義された変数\_PSP にセットされています。

また len はロックするデータのバイト数です。

id\_psp には適当な値を指定しないでください。さもないと、自分自身も common 領域をアクセスできなくなります。必ず自分自身の PSP アドレスを指定してください。

また、ロック解除は、自分自身以外のプロセスでは不可能ですので、ロックを解除せずにプログラムを終了すると、二度とロック解除できなくなりますので注意してください。

**戻 り 値** 負の数ならエラーを表します。

**参 照 関 数** COMMON\_CK, COMMON\_RD, COMMON\_WT, COMMON\_FRE,  
COMMON\_DEL

## プログラム例

```

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

/* extern int  _PSP; */

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

    if (argc == 2)
        name = argv[1];
    else {
        puts("COMMON領域の名前を指定してください。");
        exit(1);
    }

    printf("COMMON領域 %s", name);
    if ((size = COMMON_CK(name)) < 0)
        printf("は存在しません。%n");
    else if (COMMON_LK(name, 0, _PSP, size) >= 0)
        printf("%dでロックしました。%n", _PSP);
    else
        printf("のロックに失敗しました。%n");
    exit(0);
}

```

# COMMON\_RD

レベル 1

**書 式** # include <doslib. h>

```
int COMMON_RD (name, pos, buffer, len) ;
unsigned char *name; /* ブロックの名前 */
int pos; /* 読み始める位置 */
unsigned char *buffer; /* 読み出したデータを格納する領域 */
int len; /* 読み出すデータの長さ */
```

**機 能**

common 領域において、指定されたブロックよりデータを読み出します。

name にはブロックの名前を指定します。

pos にはブロックに格納されているデータを、何バイト目から読み出すか指定します。

buffer には読み出したデータを格納する領域へのポインタを指定します。

また len は読み出すバイト数です。

**戻 り 値**

実際に読み込んだバイト数を返します。

負の数ならエラーを表します。

**参 照 関 数**

COMMON\_CK, COMMON\_WT, COMMON\_LK, COMMON\_FRE,  
COMMON\_DEL

## プログラム例

```

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

#define LEN   256

main(int argc, char *argv[])
{
    char      buf[256];
    char      *name;
    int       size;

    if (argc == 2)
        name = argv[1];
    else {
        puts("COMMON領域の名前を指定してください。");
        exit(1);
    }

    printf("COMMON領域%s", name);
    if (COMMON_CK(name) < 0)
        printf("は存在しません。%n");
    else if ((size = COMMON_RD(name, 0, buf, LEN)) < 0)
        printf("の読み込みに失敗しました。%n");
    else
        printf("から%dバイト読み込みました。%n", size);
    exit(0);
}

```

# COMMON\_WT

## レベル 1

**書 式** # include <doslib. h>

```
int COMMON_WT (name, pos, buffer, len) ;
unsigned char *name; /* ブロックの名前 */
int pos; /* 書き始める位置 */
unsigned char *buffer; /* 書き込むデータの格納領域 */
int len; /* 書き込むデータの長さ */
```

**機 能** common 領域において、指定されたブロックヘータを書き込みます。  
name はブロックの名前です。  
pos はブロック中のデータ領域に書き込み始める位置を指定します。  
buffer は、書き込むデータを格納してある領域へのポインタです。  
また len は書き込むバイト数です。  
なお、len を 0 にすると、ブロックを切り詰めます。

**戻り値** 実際に書き込んだバイト数を返します。  
負の数ならエラーを表します。

**参照関数** COMMON\_CK, COMMON\_RD, COMMON\_LK, COMMON\_FRE,  
COMMON\_DEL

## プログラム例

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

#define LEN 6

static char buf[] = "X68000";

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

    if (argc == 2)
        name = argv[1];
    else {
        puts("COMMON領域の名前を指定してください。");
        exit(1);
    }

    printf("COMMON領域%s", name);
    if (COMMON_CK(name) >= 0) {
        if ((n = COMMON_WT(name, 0, buf, LEN)) >= 0)
            printf("へ%dbyte書き込みました。%n", n);
        else
            printf("への書き込みに失敗しました。%n");
    } else {
        if (COMMON_WT(name, 0, buf, LEN) >= 0)
            printf("を新たに作成しました。%n");
        else
            printf("の新規作成に失敗しました。%n");
    }
    exit(0);
}
```

# COMOUT

レベル 1

**書 式** #include <doslib.h>

```
void COMOUT (code) ;  
int code; /* 出力する文字 */
```

**機 能** RS-232C 回線へ 1 バイト出力します。

また、このときブレークチェックも行います (^C)。

**戻 り 値** 戻り値はありません。

**参 照 関 数** COMIN, COUTSNS, OUT232C

# CONSNS

レベル 1

**書 式** #include <doslib.h>

```
int CONSNS ( );
```

**機 能** 画面への出力の可・不可を調べます。

**戻 り 値** 出力不可のときには0、出力可能ならば0以外を返します。

# CONTRAST

## レベル 1

**書 式** #include <iocslib. h>

```
int CONTRAST (MODE) ;  
int MODE; /* コントラストのモード */
```

**機 能** コントラストの設定を行います。

MODE の設定値とその内容は次の通りです。

- 0~15 : 各コントラストの設定
- 1 : コントラストの通知要求をします。
- 2 : コントラストをシステムの既定値に設定します。

**戻 り 値** 設定変更前のコントラストを返します。

# COUTSNS

レベル 1

**書 式** #include <doslib. h>

```
int COUTSNS ( ) ;
```

**機 能** RS-232C 回線への出力の可・不可を調べます。

**戻 り 値** 出力不可のときには 0、出力可能ならば 0 以外を返します。

**参 照 関 数** CINSNS

# C\_PRINT

## レベル 1

**書 式** #include <doslib.h>

```
int C_PRINT (msgptr) ;  
unsigned char *msgptr ; /* 文字列格納領域へのポインタ */
```

**機 能** スル文字で終了する文字列データを表示します。  
このファンクションコールは、CON デバイスによりサポートされます。

**戻 り 値** 処理が正常に終了したときは、0 を返します。

**参 照 関 数** C\_PUTC

# C\_PUTC

レベル 1

**書 式** #include <doslib.h>

```
int C_PUTC (code) ;  
int code; /* 表示するデータ */
```

**機 能** 指定した1バイトデータを表示します。  
このファンクションコールは、CON デバイスによりサポートされます。

**戻 り 値** 処理が正常に終了したときは、0 を返します。

**参 照 関 数** C\_PRINT

# CREATE

## レベル 1

**書 式** #include <doslib.h>

```
int CREATE (file, atr) ;  
unsigned char *file; /* ファイル名格納領域へのポインタ */  
int atr; /* ファイル属性 */
```

**機 能** ファイルを新規に作成します。  
atr には、次のファイル属性を指定します。

ビット 0 : 読み込み専用	} 組み合わせで 0x00~0x07 まで指定でき ます。
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 ID	} この3つのうちどれか1つを指定して上記ビ ット 0~ビット 2 と組み合わせた値を指定し ます。
ビット 4 : ディレクトリ	
ビット 5 : 通常のファイル	

**戻 り 値** ファイルハンドルの値を返します。  
負の数なら、エラーを表します。

# C\_RIGHT

レベル 1

**書式** #include <doslib.h>

```
int C_RIGHT (n) ;
```

```
* int n; /* カーソル位置の移動桁数 */
```

**機能** カーソル位置を n 桁右へ移動します。

ただし、最右端より右へ移動しようとしても、左スクロールは行われません。

n に 0 を指定した場合、1 とみなされます。

このファンクションコールは、CON デバイスによりサポートされます。

**戻り値** 処理が正常に終了したときは、0 を返します。

**参照関数** C\_LEFT

ページ

# CRTCRAS

## レベル 0

**書 式** #include <ioclib. h>

```
int CRTCRAS (ADDRESS, LUSTER) ;
unsigned char *ADDRESS; /* 割り込み処理アドレス */
int LUSTER; /* ラスタ */
```

**機 能** CRTC に設定した割り込みラスタになったときに割り込みを行います。  
ADDRESS には割り込み処理アドレスを指定します (0 ならば割り込み禁止)。  
LUSTER にはラスタを指定します。

**戻 り 値** 割り込み可能なときは 0、すでに使用中の場合には 0 以外の値を返します。

ラスタ番号	ラスタ名	ラスタサイズ	CRTMODE =
0	high	812 × 812	0 = high
1	low	812 × 812	1 = low
2	high	328 × 328	2 = high
3	low	328 × 328	3 = low
4	high	812 × 812	4 = high
5	low	812 × 812	5 = low
6	high	328 × 328	6 = high
7	low	328 × 328	7 = low
8	high	812 × 812	8 = high
9	low	812 × 812	9 = low
10	high	328 × 328	10 = high
11	low	328 × 328	11 = low
12	high	812 × 812	12 = high
13	low	812 × 812	13 = low
14	high	328 × 328	14 = high
15	low	328 × 328	15 = low
16	high	812 × 812	16 = high

CRTMODE が 1 のときは、現在の CRT モードを返します。  
この以外の場合は割り込み可能ではありません。

**戻 り 値**

# CRTMOD

レベル 0

**書 式** #include <iocslib.h>

```
int CRTMOD (CRTMODE) ;  
int CRTMODE; /* CRT のモード */
```

**機 能** CRT のモードを指定します。

このときテキスト画面はクリアされて、表示モードとなり、テキストパレットは標準に戻ります。

グラフィック画面、およびスプライト・BG はクリアされず、無表示モードとなります。

CRTMODE に -1 を指定すると、現在の CRT モードを返します。

CRTMODE が 0x100 以上ならば、下位 8 ビットでモードの切り替えのみ行います (画面クリア、パレット・コントラスト・表示モードの初期化はしません)。

CRTMODE に指定できる値とその内容は次の通りです。

CRTMODE= ディスプレイ解像度	表示ドット数 横×縦	テキスト色/ グラフィック色	グラフィック 横ドット数
0 = high	512×512	16/16	1024
1 = low	512×512	16/16	1024
2 = high	256×256	16/16	1024
3 = low	256×256	16/16	1024
4 = high	512×512	16/16	512
5 = low	512×512	16/16	512
6 = high	256×256	16/16	512
7 = low	256×256	16/16	512
8 = high	512×512	16/256	512
9 = low	512×512	16/256	512
10 = high	256×256	16/256	512
11 = low	256×256	16/256	512
12 = high	512×512	16/65536	512
13 = low	512×512	16/65536	512
14 = high	256×256	16/65536	512
15 = low	256×256	16/65536	512
16 = high	768×512	16/16	1024

**戻り値** CRTMODE が -1 のとき、現在の CRT モードを返します。  
これ以外の場合には戻り値はありません。

# C\_UP

## レベル 1

**書 式** #include <doslib.h>

```
int C_UP (n) ;  
int n; /* カーソル位置の移動行数 */
```

**機 能**

カーソル位置を n 行上へ移動します。

ただし、先頭行より上へ移動しようとしても、スクロールダウンは行われません。

n に 0 を指定した場合、1 とみなされます。

このファンクションコールは、CON デバイスによりサポートされます。

**戻 り 値**

処理が正常に終了したときは、0 を返します。

**参 照 関 数**

C\_DOWN

# C\_UP\_S

レベル 1

**書 式** #include <doslib.h>

```
int C_UP_S ( ) ;
```

**機 能** カーソル位置を 1 行上へ移動します。

先頭行より上へ移動するときにはスクロールダウンが行われます。

このファンクションコールは、CON デバイスによりサポートされます。

**戻 り 値** 処理が正常に終了したときは、0 を返します。

**参 照 関 数** C\_DOWN\_S

# CURDIR

## レベル 1

**書 式** #include <doslib.h>

```
int CURDIR (drive, buffer) ;
int drive; /* ドライブ番号 */
unsigned char * buffer; /* パス名格納領域へのポインタ */
```

**機 能** drive で指定したドライブのカレントディレクトリを buffer が指す領域に設定します。  
drive に指定するドライブ番号は次の通りです。

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

また、buffer には次のようにパス名が設定されます。

```
" " * ルートディレクトリの場合
" TEST¥DOC" * ディレクトリ名が¥TEST¥DOC の場合
```

**戻り値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) でエラー。

# CURDRV

レベル 1

**書式** #include <doslib.h>

int CURDRV ( ) ;

**機能** カレントドライブの番号を調べます。

**戻り値** カレントドライブがどのドライブか返します。  
0 ならばドライブ A、1 ならばドライブ B となります。

## プログラム例

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

main()
{
    int drive; /* ドライブ番号 */

    drive = CURDRV() + 'A';
    printf("カレントドライブは、%cドライブです。%n", drive);
}
```

# C\_WIDTH

## レベル 1

**書 式** #include <doslib.h>

```
int C_WIDTH (n) ;  
int n; /* 画面のモード */
```

**機 能** 画面サイズの設定を行います。

このファンクションコールは、CON デバイスによりサポートされます。

なお、nで指定できるモードは次の通りです。

- 0 : 高解像度 768×512 グラフィックなし
- 1 : 高解像度 768×512 グラフィック 16 色
- 2 : 高解像度 512×512 グラフィックなし
- 3 : 高解像度 512×512 グラフィック 16 色
- 4 : 高解像度 512×512 グラフィック 256 色
- 5 : 高解像度 512×512 グラフィック 65536 色
- 1 : 現在のモードを返す

**戻り値** 設定変更前のモード (0~5) を返します。

# C\_WINDOW

レベル 1

**書 式** #include <doslib.h>

```
int C_WINDOW (ys, yl) ;  
int ys; /* 始点 Y 座標 */  
int yl; /* スクロール行数 */
```

**機 能** スクロールの範囲を設定します。

このとき、カーソルはホームポジションに移動します。

ys+yl の値は、ファンクションキー行のモードが 3 のときは 32 まで、これ以外のときは 31 までとします。

絶対座標の (0, ys) が論理座標の (0, 0) となります。

なお、このファンクションコールは、CON デバイスによりサポートされます。

**戻り値** 処理が正常に終了したときは、0 を返します。

# DAKJOB

レベル 0

**書 式** #include <iocslib.h>

```
int DAKJOB (BUFEND) ;
unsigned char *BUFEND; /* ヌル文字のアドレス */
```

**機 能** 濁点処理を行います。

BUFEND には、全角文字列の終端であるヌル文字のアドレスを指定します。最後の全角文字に濁点処理をできないときは、文字列に (\\) を追加しますので、ヌル文字の後に 3 バイト分の領域が必要です。また、(\\) を追加した場合は (\\) の後にヌル文字を付けますので、文字列の途中を指定すると (\\) の後の文字列が切れてしまいます。したがって文字列の途中を指定しないでください。

**戻 り 値** 次の内容を返します。

- 0 : 最後の全角文字に濁点処理をした
- 2 : (\\) を追加した

**参 照 関 数** HANJOB

**プログラム例**

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

static char str[13] = "かきくけこ"; /* 全角文字列バッファ */

main()
{
    char *endp; /* バッファ最終アドレス */

    printf("%s\n", str);
    endp = str + strlen(str);

    DAKJOB(endp);

    printf("%s\n", str);
}
```

# DATEASC

レベル 0

**書 式** #include <iocslib.h>

```
int DATEASC (BINDATE, BUF) ;
int BINDATE; /* 日付 (2進数) */
unsigned char *BUF; /* 文字列格納領域へのポインタ */
```

**機 能** 2進データを、日付を表す文字列に変換し、格納します。

BINDATE の形式と指定内容は次の通りです。

FFFFyyyy yyyyyyyy mmmmmmmm dddddddd

FFFF : 文字列の変換形式

0: 1987/08/07

1: 1987-08-07

2: 87/08/07

3: 87-08-07

yyyyyyyyyyyy : 年 (1980~2079)

mmmmmmmm : 月 (01~12)

ddddddd : 日 (01~31)

BUF には、日付を表す文字列の格納領域の先頭アドレスを指定します。

この領域は最低 11 バイト以上確保してください。

BUF にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

**戻 り 値** エラーが発生したときは -1 を返します。

## プログラム例

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

#define STYLE 0
#define YEAR 1990
#define MONTH 3
#define DATE 14

main()
{
    int bin; /* 日付(2進数) */
    char buf[11]; /* 文字列格納領域 */

    bin = STYLE << 28;
    bin |= YEAR << 16;
    bin |= MONTH << 8;
    bin |= DATE << 0;
    DATEASC(bin, buf);
    printf("date = %s\n", buf);
}
```

# DATEBIN

レベル 0

**書 式** #include <iocslib.h>

```
int DATEBIN (BCDDATE) ;
int BCDDATE; /* 日付 */
```

**機 能** 日付を BCD から 2 進数に変換します。

BCDDATE の形式と指定内容は次の通りです。

```
0000WWWW YYYYYYYY MMMMMMMM DDDDDDDD
```

WWWW : 曜日カウンタ (0~6) (0:日曜日)

YYYYYYYY : 年 (00~99, 1980 年からの相対年数)

MMMMMMMM : 月 (01~12)

DDDDDDDD : 日 (01~31)

上記の年, 月, 日はいずれも BCD2 桁です。

**戻 り 値** 2 進数に変換した日付を返します。日付の形式は次の通りです。

```
wwwyyyyy yyyyyyy mmmmmmm dddddd
```

www : 曜日 (0~6)

yyyyyyyyyy : 年 (1980~2079)

mmmmmmm : 月 (01~12)

ddddddd : 日 (01~31)

DATECNV

プログラム例

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

main()
{
    int    bcd;    /* 日付(BCD) */
    int    bin;    /* 日付(2進数) */

    bcd = BINDATEGET();
    bin = DATEBIN(bcd);

    printf("day   = %d\n", (bin >> 28) & 0x0f);
    printf("year  = %d\n", (bin >> 16) & 0x0fff);
    printf("month = %d\n", (bin >> 8) & 0xff);
    printf("date  = %d\n", (bin >> 0) & 0xff);
}
```

友 愛

進 進

要 求

プログラマ

00000000 00000000 00000000 00000000  
 xxxxxxxx = 日 (1980-2079)  
 yyyyyyyyy = 月 (01-12)  
 zzzzzzzzz = 日 (01-31)

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

main()
{
    int    bin;    /* 日付(2進数) */

    bin = DATECNV("1980/03/14");

    printf("year  = %d\n", (bin >> 16) & 0x0fff);
    printf("month = %d\n", (bin >> 8) & 0xff);
    printf("date  = %d\n", (bin >> 0) & 0xff);
}
```

# DATECNV

レベル 0

**書 式** #include <iocslib.h>

```
int DATECNV (ADDRESS) ;
unsigned char * ADDRESS; /* 日付を表す文字列へのポインタ */
```

**機 能**

日付を表す文字列を2進数に変換します。  
ADDRESSには、日付を表す文字列の先頭アドレスを指定します。  
ADDRESSにはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。  
また、日付データは次のように指定します。

```
"1987/08/07"
```

区切り記号は/ (スラッシュ) でも- (マイナス) でも、それら以外でもかまいません。

**戻り値**

エラーが発生したときは-1を返します。  
ただし、うるう年や大小の月の判定は行いません。  
変換を行ったときは2進数に変換した日付を返します。  
日付の形式は次の通りです。

```
0000yyyy yyyyyyyy mmmmmmmm dddddddd
```

```
yyyyyyyyyyyy = 年 (1980~2079)
```

```
mmmmmmmmm = 月 (01~12)
```

```
ddddddd = 日 (01~31)
```

**プログラム例**

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

main()
{
    int bin; /* 日付(2進数) */

    bin = DATECNV("1990/03/14");

    printf("year = %d\n", (bin >> 16) & 0x0fff);
    printf("month = %d\n", (bin >> 8) & 0xff);
    printf("date = %d\n", (bin >> 0) & 0xff);
}
```

# DAYASC

レベル 0

**書式** #include <iocslib.h>

```
void DAYASC (BINDAY, BUF) ;
int BINDAY ; /* 曜日 */
unsigned char * BUF ; /* 文字列格納領域へのポインタ */
```

**機能** 2進数を、曜日を表す文字列に変換し、格納します。

BINDAY には曜日 (0~6) を指定します。範囲外の値を指定すると、8で割った余りが使用されます (つねに0~7になります)。

BUF には、曜日を表す文字列の格納領域の先頭アドレスを指定します。この領域は最低3バイト以上確保してください。

BUF にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないように注意してください。

**戻り値** 戻り値はありません。

BUF には“日”、“月”、“火”、“水”、“木”、“金”、“土”、または“?”のいずれかがセットされます。

“?”がセットされたときは、7または8で割った余りが7になる値を指定したことを表します。

## プログラム例

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

main()
{
    char buf[3]; /* 文字列格納領域 */
    int y; /* 曜日 */

    for (y = 0; y < 7; y++) {
        DAYASC(y, buf);
        printf("曜日 = %s曜日%#n", buf);
    }
}
```

# DEFCHR

レベル 0

**書式** #include <iocslib.h>

```
int DEFCHR (TYPE, CODE, ADDRESS);  
int TYPE; /* 文字の大きさ */  
int CODE; /* 漢字コード */  
unsigned char *ADDRESS; /* 外字パターン格納領域のアドレス */
```

**機能** CODEで指定された外字に、ADDRESSで指定した外字パターンを定義します。TYPEには8または12を指定します。TYPEに0を指定した場合は8と同じです。CODEにはシフトJISコードを指定しますが、JIS漢字コード(0x7621~0x777E)を指定することもできます。TYPEに0か8を指定した場合は32バイトの外字パターンが、TYPEに12を指定した場合は72バイトの外字パターンが必要です。それぞれ、16×16ドット、24×24ドットのパターンになります。

**戻り値** 処理が正常に終了したときは0、指定した漢字コードが外字でなかったときにはエラーコード(-1)を返します。

## プログラム例

```
#include <stdio.h>  
#include <iocslib.h>  
  
main()  
{  
    char    buf[32];      /* 漢字パターン格納領域 */  
    int     type;        /* 文字の大きさ */  
    int     code;       /* 外字コード */  
    int     i;  
  
    for (i = 0; i < 32; i++)  
        buf[i] = 0xaa;  
    type = 8;  
    code = 0xebe9;  
    DEFCHR(type, code, buf);  
  
    printf("0xebe9: %c%c¥n", 0xeb, 0xe9);  
}
```

# DELETE

## レベル 1

**書 式** #include <doslib.h>

```
int DELETE (file) ;  
unsigned char *file; /* 削除するファイル名へのポインタ */
```

**機 能** file で指定するファイルを削除します。  
ワイルドカードや、ディレクトリの指定はできません。  
また、オープンされているファイルに対して実行した場合はエラーになります。

**戻 り 値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) 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は 0 以外の正の数の場合があります。

# DENSNS

レベル 0

**書 式** #include <iocslib.h>

```
void DENSNS ( );
```

**機 能** 電卓処理をします。

電卓処理は、B\_KEYINP、B\_KEYSNS 等のキー入力関数で処理されます。したがって、マウスのみで動作するプログラムのようにキー入力を行わない場合には、ソフトキーボードが表示されていても電卓が使えないことがあります。このような場合、この DENSNS を呼びながらマウス処理をしておけば、電卓が使用できるようになります。

**戻 り 値** 戻り値はありません。

# DISKRED

## レベル 1

**書 式** #include <doslib. h>

```
void DISKRED (adr, drive, sector, seclen) ;  
unsigned char *adr; /* データ読み込み領域へのポインタ */  
int drive; /* ドライブ番号 */  
int sector; /* 読み込むセクタの先頭 */  
int seclen; /* 読み込むセクタ数 */
```

**機 能** ブロックデバイスの直接入力を行います。

読み込みはセクタ単位で行うので、adr から始まるバッファのサイズは読み込みを行うデバイスの1セクタあたりのバイト数の整数倍にしてください。

なお、標準のデバイスでは1セクタは1024バイトですので、1024の倍数を指定してください。

drive は、読み込みを行うドライブ番号です。

0がカレントドライブ、1がドライブ A、2がドライブ Bとなります。

仮想ドライブや、仮想ディレクトリに割り当てた実ドライブはアクセスできません。

sector および seclen は、65535 までを指定してください。これを超えるセクタ位置やセクタ数は指定できません。

もし大容量のドライブをアクセスしたいならば、DISKRED2 関数を使用してください。

**戻り値** 戻り値はありません。

**参照関数** DISKWRT, DISKRED2, DISKWRT2

DISKRED

プログラム例

```

#include      <stdio.h>
#include      <doslib.h>

#define SECT_SIZE      1024

main()
{
    unsigned char  buf[SECT_SIZE]; /* データ格納バッファ*/
    int            drive = 1;      /* ドライブ番号 */
    int            sector = 5;     /* 読込むセクタの先頭 */
    int            seclen = 1;    /* 読み込むセクタ数 */
    int            i;

    DISKRED(buf, drive, sector, seclen);

    for (i = 0; i < 128; i++) {
        printf("%02x", buf[i]);
        printf("%c", (((i % 16) == 15) ? '\n' : ' '));
    }
}

```

左 書

添 装

装 卸

修 理 要 求

# DISKRED2

## レベル 1

**書 式** # include <doslib.h>

```
void DISKRED2 (adr, drive, sector, seclen) ;  
unsigned char *adr; /* データ読み込み領域へのポインタ */  
int drive; /* ドライブ番号 */  
long sector; /* 読み込むセクタの先頭 */  
long seclen; /* 読み込むセクタ数 */
```

**機 能** ブロックデバイスの直接入力を行います。

読み込みはセクタ単位で行うので、adr から始まるバッファのサイズは読み込みを行うデバイスの1セクタあたりのバイト数の整数倍にしてください。

なお、標準のデバイスでは1セクタは1024バイトですので、1024の倍数を指定してください。

drive は、読み込みを行うドライブ番号です。

0がカレントドライブ、1がドライブA、2がドライブBとなります。

仮想ドライブや、仮想ディレクトリに割り当てた実ドライブはアクセスできません。

DISKRED 関数に対して、DISKRED2 関数は大容量ドライブ・特殊ドライブに対応しており、sector や seclen の値を 65535 を超えるロングワードで指定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** DISKRED, DISKWRT, DISKWRT2

## プログラム例

```

#include      <stdio.h>
#include      <doslib.h>

#define SECT_SIZE      1024

main()
{
    unsigned char  buf[SECT_SIZE]; /* データ格納バッファ*/
    int            drive = 1;      /* ドライブ番号 */
    long           sector = 5L;    /* 読み込むセクタの先頭 */
    long           seclen = 1L;    /* 読み込むセクタ数 */
    int            i;

    DISKRED2(buf, drive, sector, seclen);
    for (i = 0; i < 128; i++) {
        printf("%02x", buf[i]);
        printf("%c", ((i % 16) == 15) ? '\n' : ' ');
    }
}

```

# DISKWRT

## レベル 1

**書 式** #include <doslib.h>

```
void DISKWRT (adr, drive, sector, seclen) ;  
unsigned char *adr; /* データの先頭ポインタ */  
int drive; /* ドライブ番号 */  
int sector; /* 書き出しセクタの先頭 */  
int seclen; /* 書き出しセクタ数 */
```

**機 能** ブロックデバイスの直接出力を行います。

書き出しはセクタ単位で行うので、adr から始まるバッファのサイズは書き出しを行うデバイスの1セクタあたりのバイト数の整数倍にしてください。

なお、標準のデバイスでは1セクタは1024バイトですので、1024の倍数を指定してください。

drive は、書き出しを行うドライブ番号です。

0がカレントドライブ、1がドライブ A、2がドライブ Bとなります。

仮想ドライブや、仮想ディレクトリに割り当てた実ドライブはアクセスできません。

sector および seclen は、65535 までを指定してください。これを超えるセクタ位置やセクタ数は指定できません。

もし大容量のドライブをアクセスしたいならば、DISKWRT2 関数を使用してください。

**戻り値** 戻り値はありません。

**参照関数** DISKRED, DISKRED2, DISKWRT2

### プログラム例

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

#define SECT_SIZE 1024

/* 注意! */ /* 注意! */ /* 注意! */
/* このプログラムを実行するとドライブBの内容を破壊します。 */
main()
{
    void dump(char *);
    char buf[SECT_SIZE]; /* データ格納バッファ */
    int drive = 2; /* ドライブ番号 */
    int sector = 5; /* 書き出しセクタの先頭 */
    int seclen = 1; /* 書き出しセクタ数 */
    int i;

    DISKRED(buf, drive, sector, seclen);
    dump(buf);

    for (i = 0; i < SECT_SIZE; i++)
        buf[i] = 0x40;
    DISKWRT(buf, drive, sector, seclen);

    DISKRED(buf, drive, sector, seclen);
    dump(buf);
}

static void dump(char *buf)
{
    int i;

    for (i = 0; i < SECT_SIZE; i++) {
        printf("%02x", buf[i] & 0xff);
        printf("%c", (((i % 16) == 15) ? '\n' : ' '));
    }
}
```

# DISKWRT2

## レベル 1

**書 式** # include <doslib.h>

```
void DISKWRT2 (adr, drive, sector, seclen) ;
unsigned char *adr; /* 書き出すデータの先頭ポインタ */
int drive; /* ドライブ番号 */
long sector; /* 書き出しセクタの先頭 */
long seclen; /* 書き出しセクタ数 */
```

**機 能**

ブロックデバイスの直接出力を行います。

書き出しはセクタ単位で行うので、adr から始まるバッファのサイズは書き出しを行うデバイスの1セクタあたりのバイト数の整数倍にしてください。

なお、標準のデバイスでは1セクタは1024バイトですので、1024の倍数を指定してください。

drive は、書き出しを行うドライブ番号です。

0 がカレントドライブ、1 がドライブ A、2 がドライブ B となります。

仮想ドライブや、仮想ディレクトリに割り当てた実ドライブはアクセスできません。

DISKRED 関数に対して、DISKRED2 関数は大容量ドライブ・特殊ドライブに対応しており、sector や seclen の値を 65535 を超えるロングワードで指定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** DISKWRT, DISKWRT, DISKRED2

## プログラム例

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

#define SECT_SIZE 1024

/* 注意! */ /* 注意! */ /* 注意! */
/* このプログラムを実行するとドライブBの内容を破壊します。 */
main()
{
    void dump(char *);
    char buf[SECT_SIZE]; /* データ格納バッファ */
    int drive = 2; /* ドライブ番号 */
    long sector = 5L; /* 書き出しセクタの先頭 */
    long seclen = 1L; /* 書き出しセクタ数 */
    int i;

    DISKRED2(buf, drive, sector, seclen);
    dump(buf);

    for (i = 0; i < SECT_SIZE; i++)
        buf[i] = 0x40;
    DISKWRT2(buf, drive, sector, seclen);
    DISKRED2(buf, drive, sector, seclen);
    dump(buf);
}

static void dump(char *buf)
{
    int i;
    for (i = 0; i < SECT_SIZE; i++) {
        printf("%02x", buf[i] & 0xff);
        printf("%c", (((i % 16) == 15) ? '\n' : ' '));
    }
}
```

# DMAMODE

## レベル 0

**書式** #include <iocslib, h>

```
int DMAMODE ( );
```

**機能** DMA の実行モードを調べます。

**戻り値** DMA の実行モードを返します。

```
0x00 : 何もしていない
0x8A : 転送中 (DMAMOVE 実行中)
0x8B : 転送中 (DMAMOV_L 実行中)
0x8C : 転送中 (DMAMOV_A 実行中)
```

### プログラム例

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

#define DIRECT 0x00 /* buf1 → buf2 */
#define SRCINC 0x04 /* buf1インクリメント */
#define DSTINC 0x01 /* buf2インクリメント */
#define SIZE 128 /* 転送バイト数 */

main()
{
    unsigned char buf1[SIZE]; /* 転送元 */
    unsigned char buf2[SIZE]; /* 転送先 */
    int mode; /* モード */
    int i;

    for (i = 0; i < SIZE; i++)
        buf1[i] = i;
    if (DMAMODE() == 0) {
        mode = DIRECT | SRCINC | DSTINC;
        DMAMOVE(buf1, buf2, mode, SIZE);
        for (i = 0; i < SIZE; i++) {
            printf("%02x, ", buf2[i]);
            if (i % 16 == 15)
                putchar('\n');
        }
    }
}
```

# DMAMOV\_A

レベル 0

## 書式

```
#include <iocslib.h>

void DMAMOV_A (TBLADDRESS, ADDRESS, MODE, TBLCNT) ;
struct CHAIN *TBLADDRESS; /* データチェーンテーブルのアドレス */

unsigned char *ADDRESS; /* 転送先アドレス */
int MODE; /* モード */
int TBLCNT; /* データチェーンテーブルの個数 */
struct CHAIN {
    int adr; /* 先頭アドレス */
    unsigned short len; /* 長さ */
};
```

## 機能

レイチェーンによる DMA 転送を行います。

TBLADDRESS には転送元データチェーンテーブルのアドレスを指定します。

このアドレスには、先頭アドレス、長さ、先頭アドレス、長さ、・・・の順で格納します。

ADDRESS には転送先アドレスを指定します。

TBLCNT には、転送データチェーンテーブルの個数を指定します。

MODE の DIR によって、転送先と転送元を切り換えることができます。

MODE の形式と、各項目の指定内容は次の通りです。

7      6      5      4      3      2      1      0

DIR	0	0	0	MAC	DAC
-----	---	---	---	-----	-----

### ● DIR

0 : (TBLADDRESS) から ADDRESS へ

1 : ADDRESS から (TBLADDRESS) へ

### ● MAC (TBLADDRESS)/DAC (ADDRESS)

00 : カウントしない

01 : インクリメント

10 : デクリメント

11 : 指定不可

## 戻り値

戻り値はありません。

## プログラム例

```

#include <stdio.h>
#include <iocslib.h>

#define DIR 0x00 /* tbl → dbuf */
#define MAC 0x04 /* tblインクリメント */
#define DAC 0x01 /* dbufインクリメント */
#define T_NUM 3 /* chainテーブル個数 */

main()
{
    struct CHAIN tbl[T_NUM]; /* chainテーブル */
    char sbuf[256]; /* 転送元 */
    char dbuf[100]; /* 転送先 */
    int mode; /* モード */
    int i;

    mode = DIR | MAC | DAC;
    for (i = 0; i < 255; i++)
        sbuf[i] = i;
    for (i = 0; i < T_NUM; i++) {
        tbl[i].adr = sbuf + i * 0x50;
        tbl[i].len = 0x20;
    }
    DMAMOV_A(tbl, dbuf, mode, T_NUM);
    for (i = 0; i < 96; i++) {
        printf("%02x", dbuf[i] & 0xff);
        printf("%c", (((i % 16) == 15) ? '\n' : ' '));
    }
}

```

# DMAMOVE

レベル 0

**書 式** #include <iocslib.h>

```
void DMAMOVE (ADDRESS1, ADDRESS2, MODE, BYTE) ;
unsigned char *ADDRESS1; /* 転送元アドレス */
unsigned char *ADDRESS2; /* 転送先アドレス */
int MODE; /* モード */
int BYTE; /* 転送するデータのバイト数 */
```

**機 能** DMA 転送を行います。

転送するデータのバイト数は 0xFF00 以内にしてください。

ADDRESS1 には転送元アドレス、ADDRESS2 には転送先アドレスを指定します。

MODE の DIR によって、ADDRESS1 を転送先アドレスに、ADDRESS2 を転送元アドレスに切り換えることができます。

MODE の形式と、各項目の指定内容は次の通りです。

7	6	5	4	3	2	1	0
DIR	0	0	0	MAC		DAC	

● DIR

- 0 : ADDRESS1 から ADDRESS2 へ
- 1 : ADDRESS2 から ADDRESS1 へ

● MAC (ADDRESS1)/DAC (ADDRESS2)

- 00 : カウントしない
- 01 : インクリメント
- 10 : デクリメント
- 11 : 指定不可

**戻り値** 戻り値はありません。

プログラム例

```

#include <stdio.h>
#include <iocslib.h>

#define DIRECT 0x00 /* sbuf → dbuf */
#define SRCINC 0x04 /* sbufインクリメント */
#define DSTINC 0x01 /* dbufインクリメント */
#define SIZE 128 /* 転送バイト数 */

main()
{
    unsigned char sbuf[SIZE]; /* 転送元 */
    unsigned char dbuf[SIZE]; /* 転送先 */
    int mode; /* モード */
    int i;

    for (i = 0; i < SIZE; i++)
        sbuf[i] = i;
    mode = DIRECT | SRCINC | DSTINC;
    DMAMOVE(sbuf, dbuf, mode, SIZE);
    for (i = 0; i < SIZE; i++) {
        printf("%02x", dbuf[i]);
        printf("%c", ((i % 16) == 15) ? '\n' : ' ');
    }
}

```

DIR	0	0	0	0	0
DAC					

● DIR

0 : (TRIAADDRESS) ← ADDRESS

1 : ADDRESS ← (TRIAADDRESS)

● MAC (TRIAADDRESS) ← ADDRESS

00 : ママ

01 : ト

10 : マ

11 : 指定不可

# DMAMOV\_L

レベル 0

## 書 式

```
#include <ioclib.h>

void DMAMOV_L (TBLADDRESS, ADDRESS, MODE) ;
struct CHAIN2 *TBLADDRESS; /* データチェーンテーブルのアドレス */
unsigned char *ADDRESS; /* 転送先アドレス */
int MODE; /* モード */
struct CHAIN2 {
    int adr; /* 先頭アドレス */
    unsigned short len; /* 長さ */
    struct CHAIN2 *next; /* 次のテーブルのアドレス */
};
```

## 機 能

リンクアレイチェーンによる DMA 転送を行います。

TBLADDRESS には、転送元データチェーンテーブルのアドレスを指定します。

このアドレスには、先頭アドレス・長さ・次のテーブルアドレスを格納し、次のテーブルにチェーンします。

最後のテーブルでは、次のテーブルアドレスに 0 を設定します。

ADDRESS には転送先アドレスを指定します。

MODE の DIR によって、転送先と転送元を切り換えることができます。

MODE の形式と、各項目の指定内容は次の通りです。

7          6          5          4          3          2          1          0

DIR	0	0	0	MAC	DAC
-----	---	---	---	-----	-----

- DIR
  - 0 : (TBLADDRESS) から ADDRESS へ
  - 1 : ADDRESS から (TBLADDRESS) へ
- MAC (TBLADDRESS)/DAC (ADDRESS)
  - 00 : カウントしない
  - 01 : インクリメント
  - 10 : デクリメント
  - 11 : 指定不可

## 戻 り 値

戻り値はありません。

## プログラム例

```

#include <stdio.h>
#include <ioclib.h>

#define DIRECT 0x00 /* sbuf → dbuf */
#define SRCINC 0x04 /* sbufインクリメント */
#define DSTINC 0x01 /* dbufインクリメント */
#define SIZE 256

static char sbuf[SIZE];
static char dbuf[SIZE];

struct CHAIN2 tbl3 = { /* テーブル3 */
    &sbuf[0x40],
    0x20,
    0 /* つづきなし */
};
struct CHAIN2 tbl2 = { /* テーブル2 */
    &sbuf[0x00],
    0x20,
    &tbl3 /* テーブル3へつづく */
};
struct CHAIN2 tbl1 = { /* テーブル1 */
    &sbuf[0x20],
    0x20,
    &tbl2 /* テーブル2へつづく */
};

main()
{
    int mode; /* モード */
    int i;

    for (i = 0; i < SIZE; i++)
        sbuf[i] = i;
    mode = DIRECT | SRCINC | DSTINC;
    DMAMOV_L(&tbl1, dbuf, mode);

    for (i = 0; i < 96; i++) {
        printf("%02x", dbuf[i] & 0xff);
        printf("%c", (((i % 16) == 15) ? '\n' : ' '));
    }
}

```

LED 減速	EJECT 禁止	RUFFER エラー	PROTECT エラー	NOTREADY エラー	データ 挿入
-----------	-------------	---------------	----------------	-----------------	-----------

(\*) は、modeに0を指定し、データ挿入のために有効です。  
 (\*\*) PROTECT (プロテクト) は、NOTREADY (ハット) 状態であることを示します。

# DRVCTRL

## レベル 1

**書式** #include <doslib. h>

```
int DRVCTRL (mode, drive) ;  
int mode; /* オペレーションモード */  
int drive; /* ドライブ番号 */
```

**機能** 指定ドライブの状態のセンスと設定を行います。

mode と drive の設定値は次の通りです。

● mode

- 0 : ドライブの状態のセンス (D0 のビット 7~ビット 0)
- 1 : イジェクトする (該当ドライブでオープンされているファイルがあった場合はイジェクトしない)。
- 2 : イジェクトを禁止する (mode = 1 でイジェクトを指定してもこちらの指定が優先される)。
- 3 : イジェクトを許可する (該当ドライブでオープンされているファイルはクローズされませんが、バッファは自動的にフラッシュされる)。
- 4 : ディスクがセットされていないとき、LED 点滅。
- 5 : ディスクがセットされていないとき、LED 消灯。

● drive

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

⋮

**戻り値** 指定ドライブの状態を返します。各ビットの内容は次の通りです。

7	6	5	4	3	2	1	0
LED 点滅	EJECT 禁止	BUFFER 有り	ユーザー 使用禁止	PROTECT	NOTREADY	メディア 挿入	誤挿入

(いずれもビット = 1 のとき、表記の状態であることを示します)  
なお、PROTECT (プロテクト = 1) および NOTREADY (ノットレディ = 1) は、mode に 0 を指定し、なおかつメディア挿入のときにのみ有効です。

# DRVCTRL

レベル 1

## プログラム例

```
#include <doslib.h>

#define EJECT 1 /* イジェクト */
#define CRDRV 1 /* Aドライブ */

main()
{
    int mode; /* オペレーション・モード */
    int drive; /* ドライブ番号 */

    mode = EJECT;
    drive = CRDRV;
    DRVCTRL(mode, drive);
}
```

# DRVXCHG

レベル 1

**書 式** #include <doslib.h>

```
void DRVXCHG (old, new) ;
int old; /* 旧ドライブ番号 */
int new; /* 新ドライブ番号 */
```

**機 能** old のドライブと new のドライブを入れ替えます。  
old, new の値は次のように指定します。

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

**戻り値** 戻り値はありません。

## プログラム例

```
#include <doslib.h>

main()
{
    int old; /* 旧ドライブ番号 */
    int new; /* 新ドライブ番号 */

    old = 0;
    new = 1;
    DRVXCHG(old, new);
}
```

# DSKFRE

## レベル 1

**書 式** #include <doslib.h>

```
int DSKFRE (drive, buffer) ;
int drive; /* ドライブ番号 */
struct FREEINF *buffer; /* 残り容量格納領域へのポインタ */
```

**機 能** ディスクの残り容量を調べ、その結果を buffer に格納します。  
buffer の形式は次の通りです。

```
struct FREEINF {
    unsigned short free; /* 使用可能なクラスタ数 */
    unsigned short max; /* 総クラスタ数 */
    unsigned short sec; /* 1クラスタあたりのセクタ数 */
    unsigned short byte; /* 1セクタあたりのバイト数 */
};
```

**戻り値** 使用可能なバイト数 (最大2ギガバイト) を返します。  
負の数ならエラーを表します。

### プログラム例

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

#define DRV_NO 0 /* ドライブ番号(A) */

main()
{
    struct FREEINF inf; /* 残り容量情報格納領域 */

    printf("%d byte\n", DSKFRE(DRV_NO, &inf));
    printf("free = %d\n", inf.free);
    printf("max = %d\n", inf.max);
    printf("sec = %d\n", inf.sec);
    printf("byte = %d\n", inf.byte);
}
```

# DUP

レベル 1

**書式** #include <doslib.h>

```
int DUP (fileno) ;  
int fileno; /* ファイルハンドル */
```

**機能** fileno で指定されたファイルハンドルを、新しいファイルハンドルに複写します。  
これは、標準入力、標準出力を切り替える（リダイレクトなど）を行う前に、切り替え前のファイルハンドルを保持しておくために使用します。

**戻り値** 複写された新しいファイルハンドルを返します。  
負の数ならエラーを表します。

**参照関数** DUP0, DUP2

```
#include <stdio.h>  
#include <doslib.h>  
  
#define DRV_NO 8  
  
main()  
{  
    struct PREDEF def;  
    printf("DRV_NO: %d\n", DRV_NO);  
    printf("Name: %s\n", def.name);  
    printf("max: %d\n", def.max);  
    printf("sec: %d\n", def.sec);  
    printf("type: %d\n", def.type);  
}
```

# DUPO

## レベル 1

**書式** #include <doslib.h>

```
int DUP0 (fileno, newno) ;  
int fileno; /* 複写元のファイルハンドル */  
int newno; /* 複写先のファイルハンドル */
```

**機能**

fileno のファイルハンドルを newno のファイルハンドルに強制複写します。これは、OS がデフォルトでオープンしている 0~4 までのファイルハンドルの切り替えを行う (CTTY) ためのものです。標準入出力、標準エラー出力を DUP、DUP2 など切り替えた (リダイレクトなど) あとで元にもどす場合に、CLOSE を行うことにより、元にもどります。このとき DUP0 によってどのファイルハンドルにもどるかが決定されます。newno に指定できる番号は 0~4 までです。

**戻り値**

newno の前の値を返します。  
負の数ならエラーを表します。

**参照関数**

DUP, DUP2

# DUP2

レベル 1

**書 式** #include <doslib.h>

```
int DUP2 (fileno, newno) ;  
int fileno; /* 複写元のファイルハンドル */  
int newno; /* 複写先のファイルハンドル */
```

**機 能** fileno で指定されたファイルハンドルを、newno のファイルハンドルに強制複写します。  
newno がオープンされている場合には、クローズしてから、複写します。  
これは、標準入力、標準出力を切り替えるため（リダイレクト）や、切り替えられている標準入力、標準出力を元に戻すために使用します。

**戻 り 値** 負の数ならエラーを表します。

**参 照 関 数** DUP0, DUP

## EXEC2

## レベル 1

**書式** # include <doslib. h>

```
int EXEC2 (md, fil, p1, p2) ;
int md; /* モード */
unsigned char *fil; /* ファイル名・コマンドライン等へのポインタ */
unsigned char *p1; /* コマンドライン等へのポインタ */
unsigned char **p2; /* 環境ポインタ等へのポインタ */
```

**機能** プログラムのロード・実行・モジュール番号の獲得を行います。

md は以下の通りに指定します。

ビット15 ~ 8 : モジュール番号

ビット7 ~ 0 : モード番号

md で指定したモード番号により、動作及び引数 fil, p1, p2 の意味が違います。

1) モード番号 = 0 の時

p1 で指定したコマンドラインと、p2 で指定した環境ポインタを用いて、fil で指定されたパス名のファイルをロード後実行します。

p2 の環境ポインタを 0 とすると、現在の環境と同じ環境が与えられます。

2) モード番号 = 1 の時

p1 で指定したコマンドラインと、p2 で指定した環境ポインタを用いて、fil で指定されたパス名のファイルをロードします。

p2 の環境ポインタを 0 とすると、現在の環境と同じ環境が与えられます。

3) モード番号 = 2 の時

p2 で指定した環境より path を検索し、それを元に、fil で指定したコマンド行 (コマンド名+コマンドのオプションなど) をコマンドのパス名とコマンドラインに分離し、それぞれ fil と p1 の指す領域に格納します。

fil の指す領域は 90 バイト以上、また p1 の指す領域は 256 バイト以上必要です。

p2 の環境ポインタを 0 とすると、現在の環境と同じ環境が与えられます。

モード番号 = 2 として EXEC2 関数を実行した後に、モード番号 = 0 または 1 として EXEC2 関数を実行すれば path に設定されているディレクトリにあるコマンドを簡単に実行することができます。

4) モード番号 = 3 の時

p1, p2 にそれぞれロードアドレスとリミットアドレスを指定して、fil で指定したパス名のファイルをロードします。

## 5) モード番号 = 4 の時

fil に実行アドレスを指定し、すでにロードされたプログラムを実行します。  
このモードの場合 md のモジュールは 0 とします。

## 6) モード番号 = 5 の時

fil で指定したオーバーレイ X ファイルより、pl で指定したファイル名のモジュールを検索し、見つかったらそのモジュール番号を返します。  
\* このモードの場合 md のモジュール番号は 0 とします。

モード番号 = 0, 1, 2, 3 の時、fil の最上位 8 ビットの意味は次の通りです。

- 0 : 拡張子 (.R/ .Z/ .X) にしたがって、ロードします。
- 1 : R タイプのファイルとしてロードします。
- 2 : Z タイプのファイルとしてロードします。
- 3 : X タイプのファイルとしてロードします。

**戻り値**

引数のモード番号によって異なります。

## 1) モード番号 = 0, 4 の時

戻り値が正の数ならば、子プロセスの終了コードを表します。  
負の数ならエラーを表します。  
エラーの場合ロードや実行は行われません。

## 2) モード番号 = 1 の時

戻り値が正の数ならば、ロードしたプログラムの実行アドレスを表します。  
負の数ならエラーを表します。  
エラーの場合ロードは行われません。

## 3) モード番号 = 2, 3 の時

負の数ならエラーを表します。

## 4) モード番号 = 5 の時

戻り値が正の数ならば、ビット 15~8 にモジュール番号を返したことを表します。  
負の数ならエラーを表します。

**参照関数**

EXECONLY, LOAD, LOADEXEC, LOADNLY, PATHCHK, BINDNO

# EXEONLY

レベル 1

**書 式** #include <doslib.h>

```
int EXEONLY (jmpadr) ;  
int jmpadr; /* プログラムの実行アドレス */
```

**機 能** LOAD 関数でロードしたプログラムを実行します。

**戻 り 値** 正常終了の場合には、プロセス終了コード (正の数)、異常終了したときにはエラーコード (負の数) を返します。

**参 照 関 数** LOADEXEC, LOAD, PATHCHK, LOADONLY

# EXIT

レベル 1

**書 式** #include <doslib.h>

```
void EXIT ( ) ;
```

**機 能** 現在のプロセスを終了させ、親プロセスに制御を返します。  
また、現在オープンしているファイルをすべてクローズします。  
子プロセスがオープンしたファイルについても同様です。  
親プロセスに返される終了コードは0になります。

**戻 り 値** 戻り値はありません。  
リターンせずに、親プロセスへ制御を移行します。

**参 照 関 数** EXIT2, KEEPPR

# EXIT2

## レベル 1

**書 式** #include <doslib.h>

```
void EXIT2 (code) ;
int code; /* 終了コード */
```

**機 能**

code の終了コードを持ってプログラムを終了します。

この終了コードは、親プロセスに返される終了コードになります。

現在オープンされているファイルハンドルは、すべてクローズされます。

子プロセスがオープンしたファイルハンドルについても同様です。

**戻 り 値**

戻り値はありません。

リターンせずに、親プロセスへ制御を移行します。

**参 照 関 数**

EXIT, KEEPPR

```
#include <doslib.h>
#define MAXCLUST 65535
static unsigned short aint[MAXCLUST];

main()
{
    printf("result = %d\n", PATCHK("A:VNUMBER.2YS", aint));
    printf("トランプ番号 = %d\n", aint[0]);
    printf("トランプの番 = %d\n", aint[1]);
    printf("トランプの番 = %d\n", aint[2]);
}
```

# FATCHK

レベル 1

**書 式** #include <doslib.h>

```
int FATCHK (file, buffer) ;  
unsigned char *file; /* 該当ファイルのパス名へのポインタ */  
unsigned short *buffer; /* セクタ情報格納領域へのポインタ */
```

**機 能**

指定ファイルのドライブ番号と、セクタのつながりを調べます。

これでFATが連続していることを調べるとともに、DISKRED関数、DISKRED2関数でデータ部を直接読むこともできます(戻り値が8ならば、FATは連続しています)。

なおbufferには次に示す内容が格納されます。

```
unsigned short drive; /* ドライブ番号 */  
unsigned short secno1; /* セクタの値 */  
unsigned short recen1; /* セクタ数 */  
unsigned short secno2; /* 次のセクタの値 */  
unsigned short recen2; /* 次のセクタ数 */
```

:

```
(unsigned short) 0; /* 終わりを表す0 */
```

次のセクタの値が格納されるべき所に0がセットされているならば、終わりを表します。

**戻り値**

bufferの使用バイト数(終わりを表す0も含む)を返します。  
負の数ならエラーを表します。

**参照関数**

FATCHK2

**プログラム例**

```
#include <doslib.h>  
  
#define MAXCLUST 65535  
  
static unsigned short sinf[MAXCLUST]; /* クラスタ情報 */  
  
main()  
{  
    printf("result = %d\n", FATCHK("A:\¥¥HUMAN.SYS", sinf));  
  
    printf("ドライブ番号 = %d\n", sinf[0]);  
    printf("セクタの値 = %d\n", sinf[1]);  
    printf("セクタ数 = %d\n", sinf[2]);  
}
```

# FATCHK2

## レベル 1

**書 式** # include <doslib. h>

```
int FATCHK2 (file, buffer, len) ;
unsigned char *file; /* 該当ファイルのパス名へのポインタ */
unsigned short *buffer; /* セクタ情報格納領域へのポインタ */
int len; /* バッファの最大長 */
```

**機 能**

指定ファイルのドライブ番号と、セクタの連続性を調べます。  
 FATCHK 関数に対して、FATCHK2 関数は大容量ドライブ・特殊ドライブに対応しています（戻り値が 14 ならば、FAT は連続しています）。  
 buffer には、次に示す内容が格納されます。

```
unsigned short drive; /* ドライブ番号 */
unsigned long secno1; /* 先頭のセクタ番号 */
unsigned long seclen1; /* セクタ数 */
unsigned long secno2; /* 次のセクタ番号 */
unsigned long seclen2; /* セクタ数 */
      ⋮
(unsigned long); /* 終わりを表す 0 */
```

次のセクタの値が格納されるべき所に 0 がセットされているならば終わりを表します。

**戻り値**

buffer の使用バイト数（終わりを表す 0 も含む）を返します。  
 負の数ならエラーを表します。

**参照関数**

FATCHK

## プログラム例

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

#define SIZE 1024

typedef unsigned long N_t;

main()
{
    static char buf[SIZE];
    char *inf;
    N_t s;
    N_t n;
    N_t i;
    int r;

    r = FATCHK2("/config.sys", buf, SIZE);

    inf = buf;
    printf("Drive No. = %d\n", *((unsigned short *)inf));
    for (inf += 2; (s = *((N_t *)inf)) != 0L; inf += 4) {
        inf += 4;
        n = *((N_t *)inf);
        for (i = 0; i < n; i++)
            printf("%ld ", s + i);
    }
    printf("\n");
    if (r == 14)
        printf("クラスタは連続していません。");
}
```

# FFLUSH

レベル 1

**書 式** #include <doslib.h>

```
void FFLUSH ( );
```

**機 能** ディスクのリセットを行います。

ディスクバッファの内容はすべてフラッシュしますが、ファイルのクローズはしません。

**戻 り 値** 戻り値はありません。

**参 照 関 数** flushall

# FGETC

レベル 1

**書 式** #include <doslib.h>

```
int FGETC (fileno) ;  
int fileno; /* ファイルハンドルの番号 */
```

**機 能** 指定されたファイルハンドルから入力があるまで待ち、1バイト入力します。

**戻 り 値** 入力コードを返します。

# FGETS

## レベル 1

**書 式** #include <doslib.h>

```
int FGETS (inpptr, fileno) ;
struct INPPTR { /* 入力文字列格納領域 */
    unsigned char max; /* 1行入力時の最大文字数 */
    unsigned char length; /* 1行入力が行われた文字数 */
    unsigned char buffer [256]; /* キー入力データ */
} *inpptr;
int fileno;
```

**機 能** fileno で指定されたファイルハンドルから CR コードが入力されるまで、文字列を入力します。

入力最大文字数を超えた場合は、入力最大文字数までを入力します。

length には実際に 1 行入力が行われた文字数が、buffer には入力データが格納されます。

**戻 り 値** CR コードを除いた入力文字数を返します。

**戻 り 値** GETS, GETSS

# FILEDATE

レベル 1

**書 式** #include <doslib.h>

```
int FILEDATE (fileno, datetime) ;
int fileno; /* 対象とするファイルのファイルハンドル */
int datetime; /* 日付と時刻 */
```

**機 能**

fileno で指定したファイルの日付/時刻の読み出しと設定を行います。  
datetime に 0 を指定したときは読み出しを行い、これ以外の値を指定したときには設定を行います。

datetime、および戻り値の形式は次の通りです。

datetime = YYYYYYYM MMMDDDDD hhhhhmmm mmmsssss

ビット 0~4 : (sssss) 0~29 秒 (実際の値はこれに 2 を乗ずる)  
ビット 5~10 : (mmmmmm) 0~59 分  
ビット 11~15 : (hhhhh) 0~23 時  
ビット 16~20 : (DDDDD) 1~31 日  
ビット 21~24 : (MMMM) 1~12 月  
ビット 25~31 : (YYYYYYY) 0~99 年 (1980 年から相対年数)

日付/時刻の設定を行う場合には、fileno で指定されたファイルハンドルは、書き込みが可能なモードでオープンされていなければなりません。

また、日付/時刻の設定を行ったあとで、ファイルハンドルに対して書き込みを行ってしまうと、設定された日付/時刻は無効になります。

必ずクローズを行う直前で、日付/時刻の設定を行ってください。

**戻り値**

datetime が 0 の場合のときのみ、読み出したファイルの日付/時刻を返します。  
上位 16 ビットが 0xFFFF ならばエラーを表します。

FILES

## プログラム例

```

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

main()
{
    int      fno;          /* ファイルハンドル */
    int      datetime;    /* 日付と時刻 */
    int      r;           /* 戻り値 */

    fno = open("A:\%CONFIG.SYS", O_RDONLY);

    datetime = 0;        /* 日付と時刻の読出し */
    r = FILEDATE(fno, datetime);

    printf("r      = %032b\n", r);
    printf("second = %d\n", ((r >> 0) & 0x1f) * 2);
    printf("minute  = %d\n", ((r >> 5) & 0x3f));
    printf("hour    = %d\n", ((r >> 11) & 0x1f));
    printf("day     = %d\n", ((r >> 16) & 0x1f));
    printf("month   = %d\n", ((r >> 21) & 0x0f));
    printf("year    = %d\n", ((r >> 25) & 0x7f) + 1980);
}

```

# FILES

## レベル 1

**書式** #include <doslib.h>

```
int FILES (buffer, file, atr) ;
struct FILBUF *buffer; /* ファイル情報格納領域へのポインタ */
unsigned char *file; /* ファイル名格納領域へのポインタ */
int atr; /* ファイル属性 */
struct FILBUF {
    unsigned char OS[21]; /* Human 68k が内部で使用します */
    unsigned char atr; /* ファイル属性 */
    unsigned short time; /* ファイルの時刻 */
    unsigned short date; /* ファイルの日付 */
    unsigned int filelen; /* ファイルの長さ */
    unsigned char name[23]; /* ファイル名 */
};
```

**機能** file で指定され、atr の属性をもつファイルを検索して、その情報を buffer へ格納します。

ファイル名にはワイルドカード文字が使えます。

ワイルドカードを指定したときに、最初に見つかったファイルが目的のファイルでない場合、NFILES で次のファイルを検索することができます。

atr に指定する属性は次の通りです。

- 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 ID
- 0x10 : ディレクトリ
- 0x20 : 通常ファイル

この 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対象としようとして 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。

## 戻り値

正の数（0も）で正常終了、負の数はエラーを表します。

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以外の正の数の場合があります。

## 参照関数

NFILES

## プログラム例

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

main()
{
    struct FILBUF  inf;    /* ファイル情報 */
    int            atr;    /* ファイル属性 */

    atr = 0x20;           /* 通常ファイル */

    if (FILES(&inf, " *.*", atr) >= 0) { /* 1つめがあれば */
        printf("%s\n", inf.name);        /* それを表示 */

        while (NFILES(&inf) >= 0) /* 2つめ以降がある間 */
            printf("%s\n", inf.name); /* 表示する */
    }
}
```

# FILL

レベル 0

**書 式**

```
#include <iocslib.h>
```

```
int FILL (fillptr) ;
```

```
struct FILLPTR {
```

```
    short x1; /* 始点の x 座標 */
```

```
    short y1; /* 始点の y 座標 */
```

```
    short x2; /* 終点の x 座標 */
```

```
    short y2; /* 終点の y 座標 */
```

```
    unsigned short color; /* パレットコード */
```

```
} *fillptr;
```

**機 能**

グラフィック画面にぬりつぶした四角形を描きます。

**戻 り 値**

終了コードを返します。

0 : 正常終了

-1 : グラフィックは使用不可

# FNCKEYGT

## レベル 1

**書 式** # include <doslib.h>

```
void FNCKEYGT (fno, buffer) ;
int fno; /* 定義可能キー番号 */
unsigned char *buffer; /* 読み出し用データバッファへのポインタ */
```

**機 能**

fno で指定した再定義可能キーからキーの内容を読み出します。  
このファンクションコールは、CON デバイスによりサポートされます。  
fno で指定するキー、およびデータバッファの必要サイズは次の通りです。

● fno

- 0 : すべてのキー
- 1~10 : F1~F10
- 11~20 : SHIFT+F1~F10
- 21 : ROLL UP
- 22 : ROLL DOWN
- 23 : INS
- 24 : DEL
- 25 : UP
- 26 : LEFT
- 27 : RIGHT
- 28 : DOWN
- 29 : CLR
- 30 : HELP
- 31 : HOME
- 32 : UNDO

● buffer

- |                 |                      |
|-----------------|----------------------|
| fno : 0 のとき     | 20×32+12×6 (712) バイト |
| fno : 1~20 のとき  | 32 バイト (31 バイト+¥0)   |
| fno : 21~32 のとき | 6 バイト (5 バイト+¥0)     |

**戻 り 値** 戻り値はありません。

**参 照 関 数** FNCKEYST

プログラム例

```

#include <stdio.h>
#include <doslib.h>

#define F10 10 /* F10 */
#define F20 20 /* shift+F10 */

main()
{
    char buf[32]; /* データバッファ */

    FNCKEYGT(F10, buf); /* F10読み出し */
    printf("F10: %s\n", buf + 1);

    FNCKEYGT(F20, buf); /* shift+F10読み出し */
    printf("F20: %s\n", buf);
}

```

# FNCKEYST

## レベル 1

**書 式** #include <doslib.h>

```
void FNCKEYST (fno, buffer) ;
int fno; /* 再定義可能キー番号 */
unsigned char *buffer; /* 設定用データバッファへのポインタ */
```

**機 能**

fno で指定した再定義可能キーの設定を行います (同時に、ファンクションキーの表示も行います。ただし、32 行モードのときには表示しません)。

このファンクションコールは、CON デバイスによりサポートされます。

fno で指定できるキー、およびデータバッファの必要なバイト数は次の通りです。

● fno

- 0 : すべてのキー
- 1~10 : F1~F10
- 11~20 : SHIFT+F1~F10
- 21 : ROLL UP
- 22 : ROLL DOWN
- 23 : INS
- 24 : DEL
- 25 : UP
- 26 : LEFT
- 27 : RIGHT
- 28 : DOWN
- 29 : CLR
- 30 : HELP
- 31 : HOME
- 32 : UNDO

● buffer

- fno : 0 のとき            20×32+12×6 (712) バイト
- fno : 1~20 のとき        32 バイト (31 バイト+¥0)
- fno : 21~32 のとき      6 バイト (5 バイト+¥0)

**戻り値** 戻り値はありません。

**参照関数** FNCKEYGT

# FNCKEYST

## プログラム例

```

#include      <stdio.h>
#include      <doslib.h>

#define F9    9

main()
{
    char      buf[32];          /* データバッファ */
    FNCKEYGT(F9, buf);         /* 回避 */
    FNCKEYST(F9, "FNCKEY");    /* 設定 */

    puts("キーを押すとともに戻ります。");
    while (K_KEYSNS() == 0)
        ;

    FNCKEYST(F9, buf);         /* 元に戻す */
}

```

- 0 : F1
- 1 : F2
- 2 : F3
- 3 : F4
- 4 : F5
- 5 : F6
- 6 : F7
- 7 : F8
- 8 : F9
- 9 : F10
- 10 : SHIFT-F10
- 11 : ROLL UP
- 12 : ROLL DOWN
- 13 : INS
- 14 : DEL
- 15 : CLR
- 16 : LEFT
- 17 : RIGHT
- 18 : DOWN
- 19 : CLR
- 20 : HELP
- 21 : HOME
- 22 : UNDO

● buffer

```

ino : 0 @ 3
ino : 1-20 @ 4
ino : 21-32 @ 5

```

戻り値

登録関数

# FNTGET

## レベル 0

**書 式** #include <iocslib.h>

```
int FNTGET (TYPE, CODE, BUF) ;  
int TYPE; /* 文字の大きさ */  
int CODE; /* 漢字コード */  
struct FNTBUF *BUF /* データバッファアドレス */
```

**機 能** CODEで指定された漢字パターンを、BUFで指定されたアドレスへ読み込みます。

BUFにはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないように注意してください。

TYPEには8または12を指定します。

0を指定した場合は8と同じです。

CODEにはシフトJIS漢字コードを指定しますが、JIS漢字コード(\$2121~\$7E7E)を指定することもできます。

BUFには、データバッファの先頭アドレスを指定します。

この領域は、TYPEに8または0を指定した時には36バイトの、TYPEに12を指定した時には76バイトの領域が必要です。

構造体FNTBUFのデータサイズを越える場合には、メモリ確保関数mallocなどで確保した領域を使用してください。

BUFには、漢字パターンのX方向のドット数(short)、Y方向のドット数(short)、およびパターンデータが格納されます。

**戻り値** 戻り値はありません。

例

プログラム例

```

#include      <stdio.h>
#include      <iocslib.h>

static struct FNTBUF  fnt;  /* データバッファ */

main()
{
    int  type;          /* 文字の大きさ */
    int  code;         /* 漢字コード */

    type = 8;
    code = 0x989f;

    FNTGET(type, code, &fnt);

    printf("x: %d dots\n", fnt.xl);
    printf("y: %d dots\n", fnt.yl);

    code = 0xebe9;
    DEFCHR(type, code, fnt.buffer);
    printf("%c%c\n", 0xeb, 0xe9);
}

```

先

進

戻り値

# FPUTC

## レベル 1

**書 式** #include <doslib.h>

```
void FPUTC (code, fileno) ;  
int code; /* 出力する文字コード */  
int fileno; /* 出力するファイルハンドル */
```

**機 能** 指定した文字コードをファイルハンドルに出力します。  
出力デバイスがキャラクタデバイスの場合には、ブレークチェックを行います  
( $\hat{C}:\hat{S}:\hat{P}:\hat{N}:$ )。

**戻 り 値** 戻り値はありません。

# FPUTS

レベル 1

**書 式** #include <doslib. h>

```
void FPUTS (msgptr, fileno) ;  
unsigned char *msgptr; /* 出力する文字列へのポインタ */  
int fileno; /* 出力するファイルハンドル */
```

**機 能** スル文字で終了する文字列を、ファイルハンドルに出力します。  
出力するデバイスがキャラクタデバイスの場合には、ブレークチェックを行います (^C : ^S : ^P : ^N : )。

**戻 り 値** 戻り値はありません。

# G\_CLR\_ON

レベル 0

**書式** #include <ioclib. h>

void G\_CLR\_ON ( ) ;

**機能** グラフィック画面をクリアして、表示モードにします。  
グラフィックパレットは初期状態に戻ります。

**戻り値** 戻り値はありません。

**論理異**

0x40: 割り当て済み領域  
0x50: 拡張メモリ割り当て  
0x60: 拡張メモリ割り当て

**関連関数**

MAKEASIGN, RASIGN

# GETASSIGN

レベル 1

**書 式** # include <doslib. h>

```
int GETASSIGN (buff1, buff2) ;  
unsigned char *buff1; /* ドライブ名 */  
unsigned char *buff2; /* ディレクトリ名を格納するポインタ */
```

**機 能**

仮想ドライブ、仮想ディレクトリの割り当て状況を取得します。  
戻り値によって、以下のような割り当て状況がbuff2へセットされます。

- 戻り値が0x40のとき (割り当てが存在しない)  
buff1で指定されたドライブのカレントディレクトリがbuff2へセットされます。
- 戻り値が0x50のとき (仮想ドライブの割り当て)  
buff1をアクセスしたとき、実際にアクセスされるディレクトリがbuff2へセットされます。
- 戻り値が0x60のとき (仮想ディレクトリの割り当て)  
buff1を実際にアクセスしようとしたとき、アクセスしなければならないディレクトリがbuff2にセットされます。

buff1に、MAKEASSIGN関数での1番目のパラメータと同じ内容のパラメータを指定すると、2番目のパラメータで指定していた内容がbuff2にセットされます。

**戻り値**

以下に示す割り当てモードを返します。

- 0x40: 割り当てが存在しない
- 0x50: 仮想ドライブの割り当て
- 0x60: 仮想ディレクトリの割り当て

負の数ならエラーを表します。

**参照関数**

MAKEASSIGN, RASSIGN

# GETC

## レベル 1

**書式** #include <doslib. h>

```
int GETC ( ) ;
```

**機能** 標準入力から1文字読み込みます。  
このときブレークチェックを行います (^C:^P:^N:)

**戻り値** 読み込んだ文字コードを返します。

**参照関数** INKEY, GETCHAR, getc

# GETCHAR

レベル 1

**書 式** #include <doslib. h>

```
int GETCHAR ( ) ;
```

**機 能** 標準入力から1文字読み込み、読み込んだ文字を標準出力にエコーバックします。このときブレークチェックも行います (^C : ^P : ^N: )。

**戻 り 値** 読み込んだ文字コードを返します。

**参 照 関 数** PUTCHAR, GETC, INPOUT, INKEY, getche

# GETDATE

レベル 1

**書式** #include <doslib.h>

int GETDATE ( );

**機能** 現在の日付を調べます。**戻り値** 現在の日付を返します。  
形式は次の通りです。

00000000 00000www yyyyyyyymm mmmddddd

- ビット 0~4 (dddd) : 日 (1~31)
- ビット 5~8 (mmmm) : 月 (1~12)
- ビット 9~15 (yyyyyy) : 年 (0~99, 1980年からの相対年数)
- ビット16~18 (www) : 曜日 (0 = 日・・・6 = 土)

**参照関数** GETTIM2, SETTIM2, SETDATE, GETTIME, SETTIME**プログラム例**

```

#include      <stdio.h>
#include      <doslib.h>

main()
{
    int  date;
    int  yy;
    int  mm;
    int  dd;

    date = GETDATE();

    yy = ((date >> 9) & 0x7f) + 1980;
    mm = ((date >> 5) & 0x0f);
    dd = ((date >> 0) & 0x1f);
    printf("%d年%d月%d日¥n", yy, mm, dd);
}

```

# GETDPB

レベル 1

**書式** #include <doslib.h>

```
int GETDPB (drive, dpbptr) ;
int drive; /* ドライブ番号 */
struct DPBPTR *dpbptr; /* ドライブパラメータブロック格納領域 */
```

**機能** 指定したドライブ番号のドライブパラメータブロックを指定した格納領域へ格納します。

driveに指定できるデバイス番号、ドライブパラメータブロックの形式は、それぞれ次の通りです。

● ドライブ番号

- 0 : カレントドライブ
- 1 : ドライブA
- 2 : ドライブB
- ⋮

● ドライブパラメータブロック

```
struct DPBPTR {
unsigned char drive; /* ドライブ番号 (0 = A, 1 = B) */
unsigned char unit; /* デバイスドライブで使うユニット番号 */
unsigned short byte; /* 1セクタ数あたりのバイト数 */
unsigned char sec; /* 1クラスタあたりのセクタ数-1 */
unsigned char shift; /* クラスタ、セクタ間のシフト数 */
unsigned short fatsec; /* FATの先頭セクタ番号 */
unsigned char fatcount; /* FAT領域の個数 */
unsigned char fatlen; /* FATの占めるセクタ数 (コピー分は別) */
unsigned short dircount; /* ルートディレクトリの個数 */
unsigned short datasec; /* データ部先頭のセクタ番号 */
unsigned short maxfat; /* 総クラスタ数+1 */
unsigned short dirsec; /* ルートディレクトリの先頭セクタ番号 */
int driver; /* デバイスドライブへのポインタ */
unsigned char id; /* メディアバイト */
unsigned char flg; /* dpb使用フラグ (-1のときアクセスなし) */
struct DPBPTR *next; /* 次の dpb へのポインタ */
};
```

```

unsigned short dirfat; /* カレントディレクトリのクラスタ番号 */
/* (0のときはルートディレクトリを表す) */
unsigned char dirbuf[64]; /* カレントディレクトリの文字バッファ */
};

```

**戻り値**

負の数ならエラーを表します。

**プログラム例**

```

#include <stdio.h>
#include <doslib.h>

main()
{
    struct DPBPTR dpb; /* ドライブパラメータブロック */
    int drive; /* ドライブ番号 */

    drive = 0;

    GETDPB(drive, &dpb);

    printf("drive: %d\n", dpb.drive);
    printf("unit : %d\n", dpb.unit);
    printf("byte : %d\n", dpb.byte);
    printf("sec : %d\n", dpb.sec);
}

```

# GETENV

レベル 1

## 書式

```
#include <doslib.h>
```

```
int GETENV (name, env, buffer) ;
```

```
unsigned char *name; /* 環境変数へのポインタ */
```

```
unsigned char *env; /* 環境のポインタ */
```

```
unsigned char *buffer; /* 環境変数格納領域へのポインタ */
```

## 機能

env で指定された環境から、name で指定した環境変数の内容を取り出します。  
env に 0 を指定すると、親プロセスの環境から取り出します。

環境変数の内容は buffer に格納されますが、この領域は 256 バイト必要です。

## 戻り値

負の数ならエラーを表します。

## 参照関数

SETENV

## プログラム例

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

main()
{
    char *name; /* 環境変数 */
    char *env = NULL; /* 環境 */
    char buf[256]; /* 環境変数格納領域 */

    name = "path";

    GETENV(name, env, buf);

    printf("%s\n", buf);
}
```

# GETGRM

レベル 0

**書 式** #include <ioclib.h>

```
int GETGRM (getptr) ;
struct GETPTR {
    short x1; /* 始点の X 座標 */
    short y1; /* 始点の Y 座標 */
    short x2; /* 終点の X 座標 */
    short y2; /* 終点の Y 座標 */
    unsigned char *buf_start; /* バッファ先頭アドレス */
    unsigned char *buf_end; /* バッファ終了アドレス */
} *getptr;
```

**機 能**

グラフィック画面からの読み込みを行います。

バッファには、データがパックされた型で格納されます。

**戻 り 値**

終了コードを返します。

0 : 正常終了

-1 : グラフィックは使用不可

# GETPDB

レベル 1

書 式

```
# include <doslib.h>

struct PDBADR *GETPDB ( );

struct PDBADR {
    unsigned int env; /* プロセスに与えられた環境のアドレス */
    unsigned int exit; /* プロセスが終了したさいの戻りアドレス */
    unsigned int ctrlc; /* プロセスが CTRL+C により中断された際の戻り
        アドレス */
    unsigned int errexit; /* プロセスがエラーにより中止された際の戻りア
        ドレス */
    unsigned int comline; /* プロセスに与えられたコマンドラインのアドレ
        ス */
    unsigned char handle[12]; /* プロセスのファイルハンドラの使用状況
        */
    unsigned int bss; /* プロセスの bss の先頭アドレス */
    unsigned int heap; /* プロセスのヒープの先頭アドレス */
    unsigned int stack; /* プロセスの初期スタックアドレス */
    unsigned int usp; /* 親プロセスの USP レジスタの値 */
    unsigned int ssp; /* 親プロセスの SSP レジスタの値 */
    unsigned short sr; /* 親プロセスの SR レジスタの値 */
    unsigned short abort_sr; /* アボート (中止) のときの SR レジスタの値
        */
    unsigned int abort_ssp; /* アボート (中止) のときの SSP レジスタの
        値 */
    unsigned int trap10; /* TRAP10 */
    unsigned int trap11; /* TRAP11 */
    unsigned int trap12; /* TRAP12 */
    unsigned int trap13; /* TRAP13 */
    unsigned int trap14; /* TRAP14 */
    unsigned int osflag; /* プロセスのフラグ、0で親あり、-1で OS から
        起動 */
    unsigned char reserve[28]; /* 未使用 */
    unsigned char exe_path[68]; /* exec されたファイルのパス名 */
    unsigned char exe_name[24]; /* exec されたファイル名 */
};
```

GETPR

レベル1

**機能** 現在のプロセスを表す pdbadr を求めます。

pdbadr はプログラムの先頭アドレス-0xF0 です。

なお、求めた pdbadr は SETPDB 関数で使用します。

**戻り値** 現在のプロセスを表す pdbadr の値を返します。

**参照関数** SETPDB

謝

# GET\_PR

レベル 1

書 式

```
# include <doslib.h>

int GET_PR (id, buff);

int id; /* スレッドの ID */
struct PRCPTR { /* スレッド管理情報格納領域 */
    struct PRCPTR * next_ptr; /* 次の管理領域へのポインタ */
    unsigned char wait_flg; /* normal = $00 / wait = $ff */
    unsigned char counter; /* 割り込みの度に減算される */
    unsigned char max_counter; /* counter のプリセット値 */
    unsigned char doscmd;
    unsigned int psp_id; /* プロセス ID */
    unsigned int usp_reg; /* usp 格納領域 */
    unsigned int d_reg [8]; /* データレジスタ格納領域 */
    unsigned int a_reg [7]; /* アドレスレジスタ格納領域 */
    unsigned short sr_reg; /* sr 格納領域 */
    unsigned int pc_reg; /* pc 格納領域 */
    unsigned int ssp_reg; /* ssp 格納領域 */
    unsigned short indosf;
    unsigned int indosp;
    struct PRCCTRL * buf_ptr; /* タスク間通信バッファへのポインタ */
    unsigned char name [16]; /* スレッドの名前 */
    long wait_time; /* 待ち時間の残り (ミリ秒) */
} *buff;
```

機 能

指定されたIDのバックグラウンドタスクの管理情報をbuffにコピーします。

idに-1を指定し、buff->nameにバックグラウンドタスクの名前を指定してGET\_PR関数を呼び出すと、戻り値としてバックグラウンドタスクのidが返り、管理情報がbuffにコピーされます。

idに-2を指定すると、自分自身のIDが戻り値として返り、管理情報がbuffにコピーされます。

**戻り値**

求めるバックグラウンドタスクのidを返します。

戻り値が負の数の場合はエラーコードを示します。

指定したidがおかしい場合は、0xFFFF00?? (??は不定の16進数を表します) を返します。

0x??がidの最大値です。

指定した名前のバックグラウンドタスクが存在しない場合は-1を返します。

**参照関数**

OPEN\_PR, KILL\_PR, SUSPEND\_PR, SLEEP\_PR, SEND\_PR, TIME\_PR, CHANGE\_PR

# GETS

レベル 1

**書 式**

```
#include <doslib.h>
```

```
int GETS (inpptr) ;  
    struct INPPTR { /* 入力文字列格納領域 */  
        unsigned char max ; /* 入力時の最大文字数 */  
        unsigned char length ; /* 入力が行われた文字数 */  
        unsigned char buffer [256] ; /* 入力データ */  
    } *inpptr ;
```

**機 能**

CR コードが入力データに現れるまで、文字列を入力します。

このときブレイクチェックも行います (^C : ^P : ^N :)。

入力文字数が最大文字数 inpptr->max を超えた場合は、警告音を出しますが終了はしません。

length には実際に入力が行われた文字数が、buffer には入力データが格納されます。

**戻 り 値**

CR コードを除く入力文字数を返します。

**参 照 関 数**

PRINT, GETSS, FGETS

# GETSS

## レベル 1

**書式** #include <doslib.h>

```
int GETSS (inpptr) ;
    struct INPPTR { /* 入力文字列格納領域 */
        unsigned char max; /* 入力時の最大文字数 */
        unsigned char length; /* 入力が行われた文字数 */
        unsigned char buffer [256]; /* 入力データ */
    } *inpptr;
```

**機能** CR コードが入力データに現れるまで、文字列を入力します。  
ただし、ブレイクチェックは行いません。  
また、VOID/NEWLINEで改行しません。

入力文字数が入力最大文字数 `inpmax` を超えた場合は警告音を出しますが、終了はしません。

`length` には実際に入力が行われた文字数が、`buffer` には入力データが格納されます。

**戻り値** CR コードを除いた入力文字数を返します。

**参照関数** GETS, FGETS

```

#include <stdio.h>
#include <doslib.h>

int main()
{
    int time; /* 時刻の表示 */
    int hh;
    int mm;
    int ss;

    time = GETTIME();

    hh = (time >> 10) & 0xFF;
    mm = (time >> 2) & 0xFF;
    ss = (time >> 0) & 0xFF;

    printf("時刻は %02d:%02d:%02d\n", hh, mm, ss);
}

```

# GETTIM2

レベル 1

**書 式** #include <doslib.h>

int GETTIM2 ( );

**機 能** 現在の時刻を調べます。

**戻 り 値** 現在の時刻を返します。  
形式は次の通りです。

00000000 000hhhhh 00mmmmmm 00sssss

ビット 0~5 (sssss) : 秒 (0~59)

ビット 8~13 (mmmmm) : 分 (0~59)

ビット 16~20 (hhhhh) : 時 (0~23)

**参 照 関 数** SETTIM2, GETDATE, SETDATE, GETTIME, SETTIME

## プログラム例

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

main()
{
    int    time;    /* 現在の時刻 */
    int    hh;
    int    mm;
    int    ss;

    time = GETTIM2();

    hh = (time >> 16) & 0x1f;
    mm = (time >> 8) & 0x3f;
    ss = (time >> 0) & 0x3f;
    printf("%d時%d分%d秒\n", hh, mm, ss);
}
```

# GETTIME

レベル 1

**書式** #include <doslib.h>

```
int GETTIME ( );
```

**機能** 現在の時刻を調べます。

**戻り値** 現在の時刻を返します。

形式は次の通りです。

```
hhhhmmmm mmmsssss
```

ビット 0~4 (sssss) : 秒 (0~29, 実際の値は 2 を乗ずる)

ビット 5~10 (mmmmmm) : 分 (0~59)

ビット 11~15 (hhhhh) : 時 (0~23)

**参照関数** GETTIM2, SETTIM2, GETDATE, SETDATE, SETTIME

**プログラム例**

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

main()
{
    int time; /* 現在の時刻 */
    int hh;
    int mm;
    int ss;

    time = GETTIME();

    hh = ((time >> 11) & 0x1f);
    mm = ((time >> 5) & 0x3f);
    ss = ((time >> 0) & 0x1f) * 2;
    printf("%d時%d分%d秒\n", hh, mm, ss);
}
```

# GPALET

レベル 0

**書式** #include <iocslib.h>

```
int GPALET (PALET_NO, COLOR_CODE);  
int PALET_NO; /* パレット番号 */  
int COLOR_CODE; /* カラーコード */
```

**機能**

グラフィックのパレットを設定します。  
PALET\_NO にはパレット番号 (0~15) / (0~255) / (0~65535) を指定します。  
COLOR\_CODE にはカラーコード (0~65535) を指定します。  
-1 を指定するとパレット番号の読み出しのみ行います。

**戻り値**

COLOR\_CODE に -1 を指定したときのみ、現在のカラーコードを返します。

**プログラム例**

```
#include <stdio.h>  
#include <iocslib.h>  
  
#define PALET 12  
#define COLOR 216  
  
main()  
{  
    int palet_no; /* パレット番号 */  
    int color_code; /* カラーコード */  
  
    palet_no = PALET;  
    color_code = COLOR;  
  
    G_CLR_ON();  
  
    printf("変更前: %d\n", GPALET(palet_no, -1));  
    GPALET(palet_no, color_code);  
    printf("変更後: %d\n", GPALET(palet_no, -1));  
  
    G_CLR_ON();  
}
```

## HANJOB

## レベル 0

**書 式** #include <iocslib.h>

```
int HANJOB (BUFEND) ;
unsigned char *BUFEND; /* ヌル文字のアドレス */
```

**機 能** 半濁点処理を行います。

BUFEND には、全角文字列の終端にあるヌル文字のアドレスを指定します。

最後の全角文字に半濁点処理をできない時は、文字列に (°) を追加しますので、ヌル文字の後に3バイト分の領域が必要です。

また、(°) を追加した場合は (°) の後にヌル文字を付けますので、文字列の途中を指定すると (°) の後の文字列が切れてしまいます。

したがって、文字列の途中を指定しないでください。

**戻 り 値** 次の内容を返します。

0 : 最後の全角文字に半濁点処理をした

2 : (°) を追加した

**参 照 関 数** DAKJOB

## プログラム例

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

static char str[13] = "はひふへほ"; /* 全角文字列バッファ */

main()
{
    char *endp; /* バッファ最終アドレス */

    printf("%s\n", str);
    endp = str + strlen(str);

    HANJOB(endp);

    printf("%s\n", str);
}
```

# HENDSPIC

レベル 1

**書式** #include <doslib.h>

```
void HENDSPIC (position) ;
int position; /* ウィンドウ上の表示位置 */
```

**機能** position 以降の内容をもとに戻します。  
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻り値** 戻り値はありません。

**参照関数** HENDSPIO, HENDSPIP, HENDSPIR

```

#include <stdio.h>
#include <doslib.h>

static char str[13] = "はひふへほ";

main()
{
    char *endp;
    printf("str: %s", str);
    endp = str + strlen(str);
    HANJOB(endp);
    printf("str: %s", str);
}

```

# HENDSPIO

レベル 1

**書 式** #include <doslib. h>

int HENDSPIO ( ) ;

**機 能** 入力ウィンドウをオープンします。

このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻 り 値** ウィンドウの最大表示文字数を返します。

**参 照 関 数** HENDSPIC, HENDSPIP, HENDSPIR

# HENDSPIP

レベル 1

**書 式** #include <doslib. h>

```
int HENDSPIP (position, msgptr) ;
int position; /* ウィンドウ上の表示位置 */
unsigned char *msgptr; /* メッセージ格納領域へのポインタ */
```

**機 能** メッセージを、入力ウィンドウにノーマルで表示します。  
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻 り 値** ウィンドウでの次の表示位置を返します。

**参 照 関 数** HENDSPIC, HENDSPIO, HENDSPIR

# HENDSPIR

レベル 1

**書 式** #include <doslib.h>

```
int HENDSPIR (position, msgptr) ;  
int position; /* ウィンドウ上の表示位置 */  
unsigned char *msgptr; /* メッセージ格納領域へのポインタ */
```

**機 能** メッセージを入力ウィンドウにリバースで表示します。  
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻 り 値** ウィンドウでの次の表示位置を返します。

**参 照 関 数** HENDSPIC, HENDSPIO, HENDSPIP

# HENDSPMC

レベル 1

**書 式** #include <doslib.h>

```
void HENDSPMC ( );
```

**機 能** モード表示ウィンドウをクローズします。  
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻り値** 戻り値はありません。

**参照関数** HENDSPMO, HENDSPMP, HENDSPMR

# HENDSPMO

## レベル 1

**書 式** #include <doslib.h>

```
int HENDSPMO ( ) ;
```

**機 能** モード表示ウィンドウをオープンします。

このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻 り 値** ウィンドウでの最大表示文字数を返します。

**参 照 関 数** HENDSPMC, HENDSPMP, HENDSPMR

# HENDSPMP

レベル 1

**書 式** #include <doslib.h>

```
int HENDSPMP (position, msgptr) ;  
int position; /* ウィンドウ上の表示位置 */  
unsigned char *msgptr; /* メッセージ格納領域へのポインタ */
```

**機 能**

メッセージを、モード表示ウィンドウにノーマルで表示します。

このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻 り 値**

ウィンドウ上の次の表示位置を返します。

**参 照 関 数**

HENDSPMC, HENDSPMO, HENDSPMR

# HENDSPMR

## レベル 1

**書 式** #include <doslib.h>

```
int HENDSPMR (position, msgptr) ;  
int position; /* ウィンドウ上の表示位置 */  
unsigned char *msgptr; /* メッセージ格納領域へのポインタ */
```

**機 能**

メッセージを、モード表示ウィンドウにリバースで表示します。  
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻 り 値**

ウィンドウ上の次の表示位置を返します。

**参 照 関 数**

HENDSPMC, HENDSPMO, HENDSPMP

# HENDSPSC

レベル 1

**書 式** #include <doslib. h>

```
void HENDSPSC ( ) ;
```

**機 能** 候補ウィンドウをクローズします。

このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻 り 値** 戻り値はありません。

**参 照 関 数** HENDSPSO, HENDSPSP, HENDSPSR

# HENDSPSO

## レベル 1

**書 式** #include <doslib.h>

```
int HENDSPSO ( ) ;
```

**機 能** 候補ウィンドウをオープンします。

このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻 り 値** ウィンドウの最大表示文字数を返します。

**参 照 関 数** HENDSPSC, HENDSPSP, HENDSPSR

# HENDSPSP

レベル 1

**書 式** #include <doslib.h>

```
int HENDSPSP (position, msgptr) ;  
int position; /* ウィンドウ上の表示位置 */  
unsigned char *msgptr; /* メッセージ格納領域へのポインタ */
```

**機 能** メッセージを、候補ウィンドウにノーマルで表示します。

このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻り値** ウィンドウ上の次の表示位置を返します。

**参照関数** HENDSPSC, HENDSPSO, HENDSPSR

# HENDSPSR

レベル 1

**書式** #include <doslib.h>

```
int HENDSPSR (position, msgptr) ;  
int position; /* ウィンドウ上の表示位置 */  
unsigned char *msgptr; /* メッセージ格納領域へのポインタ */
```

**機能** メッセージを、候補ウィンドウにリバーズで表示します。  
このファンクションコールは、漢字変換フロントプロセッサのためのものであり、ふつうのアプリケーションプログラムで使用してはいけません。

**戻り値** ウィンドウ上の次の表示位置を返します。

**参照関数** HENDSPSC, HENDSPSO, HENDSPSP

# HOME

レベル 0

**書 式** #include <iocslib. h>

```
int HOME (PAGE, X, Y) ;  
int PAGE; /* ページ指定 */  
int X; /* X座標 */  
int Y; /* Y座標 */
```

**機 能**

グラフィック画面の表示位置を設定します。  
設定するページはPAGEのビット0～ビット3で指定します（たとえば、ページ0を指定するときはビット0をONにします）。  
ビット0～ビット3がすべて0ならば、現在のモードで有効なページがすべて設定されます。  
XにはX座標、YにはY座標を指定します。  
ただし指定できるのはいずれも0～1023までです。

**戻り値**

終了コードを返します。

- 0 : 正常終了
- 1 : グラフィックは使用不可
- 2 : 引数が指定通りになっていない
- 3 : 現在のモードでは設定不可

# HSVTORGB

レベル 0

**書 式** #include <ioclib. h>

```
int HSVTORGB (H, S, V) ;
int H; /* カラー */
int S; /* 白レベル */
int V; /* 黒レベル */
```

**機 能** HSV 方式から RGB 方式に変換します。

H で指定できるカラーは次の通りです。

なお、それぞれの間は 32 段階に分かれていますので色合いが指定できます。

0~31 : 赤~黄色  
 32~63 : 黄色~緑  
 64~95 : 緑~シアン  
 96~127 : シアン~青  
 128~159 : 青~マゼンダ  
 160~191 : マゼンダ~赤  
 192 以上 : 禁止

S は白レベル (0 で白、0x1F で原色)、V は黒レベル (0 で黒、0x1F で原色) を示します。

また RGB はパレットで指定するデータ形式になります (緑・赤・青はそれぞれ 5 ビット。最下位ビットは 0)。

**戻 り 値** RGB を次の形式で返します。

```
00000000 00000000 GGGGRRR RRBBBB0
```

# HSYNCST

レベル 0

**書 式** #include <iocslib.h>

```
int HSYNCST (ADDRESS) ;
unsigned char *ADDRESS ; /* 割り込み処理アドレス */
```

**機 能** H-SYNC による割り込みを制御します。  
ADDRESS には割り込み処理アドレスを指定します (0 ならば割り込み禁止)。

**戻 り 値** 割り込みが可能なのは 0、すでに使用中の場合には 0 以外の値を返します。

## INDOSFLG

## レベル 1

**書式** # include <doslib. h>

int INDOSFLG ( ) ;

**機能** OS のワーク INDOS\_FLG のアドレスを返します。

スーパーバイザ領域なのでユーザーモードではアクセスできないので注意してください。

また、このアドレスが指すワークエリアに続いて、OS にとって重要な情報が格納されているので、読み出すのは構いませんが、絶対に書き直さないでください。

● ライブラリ関数 INDOSFLG ( ) の指す内容について

書式	内容
unsigned short indosf	OS 実行中レベル
unsigned char doscmd	OS 実行中ファンクション番号
unsigned char fat_flg	FAT 検索モード 0=標準 0≠先頭から
unsigned short retry_count	I/O リトライカウント 標準で 3 回
unsigned short retry_time	I/O リトライ待ち時間 標準 100 (1 秒)
unsigned short verifyf	ベリファイモード 0=オフ 0≠オン
unsigned char breakf	ブレイクモード 0=オフ、1=オン
unsigned char ctrlpf	CTRL+P モード 0=オフ 0≠オン
unsigned char reserved	システムで使用
unsigned char wkcurdrv	カレントドライブ A:=0

**戻り値** INDOS\_FLG が格納されているアドレスを返します。

# INIT\_PRN

レベル 0

**書 式** #include <iocslib.h>

```
int INIT_PRN (LINE, WIDTH) ;
int LINE; /* 印字可能行数 */
int WIDTH; /* 印字可能桁数 */
```

**機 能** プリンタポートを初期化します。

LINE には 1 ページの行数-1 を指定します (OUTPRN 関数用。0xFF でページ指定なし)。

WIDTH に 1 行の桁数-1 を指定します (OUTPRN 関数用。0xFF で行指定なし)。

**戻 り 値** 0x000020 ならばプリンタへの出力は可能です。  
0x000000 のときには出力できません。

## プログラム例

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

#define MAXLINE 66 /* 印字可能行数 */
#define MAXWIDTH 132 /* 印字可能文字数 */

main()
{
    char *str = "hello¥r¥n";
    if (INIT_PRN(MAXLINE - 1, MAXWIDTH - 1) == 0) {
        printf("プリンタへの出力はできません¥n");
        exit(1);
    }
    if (SNSPRN() == 0) {
        printf("プリンタへの出力はできません¥n");
        exit(1);
    }
    while (*str)
        OUTLPT(*str++);
}
```

# INKEY

## レベル 1

**書 式** #include <doslib.h>

```
int INKEY ( ) ;
```

**機 能** キーが押下されるまで待ち、文字コードを入力します。  
ブレークチェックは行いません。

**戻 り 値** 入力された文字コードを返します。

**参 照 関 数** GETC, GETCHAR

# INP232C

レベル 0

**書 式** #include <iocslib.h>

int INP232C ( ) ;

**機 能** RS-232C 受信データを獲得します。

**戻 り 値** RS-232C 受信データを 1 バイト返します。

## プログラム例

```
#include <iocslib.h>

main()
{
    int data; /* 受信データ */
    if ((data = ISNS232C()) != 0)
        data = INP232C();
}
```

# INPOUT

## レベル 1

**書 式** #include <doslib.h>

```
int INPOUT (code) ;
int code; /* ファンクションコード */
```

**機 能** コンソールの直接入出力を行います。  
code が 0xFF ならばキー入力を行います。  
このとき、入力待ちは行いません。  
code が 0xFE ならばキーセンスを行い、入力したキーコードを返します。  
これ以外の場合には、code を文字コードとして出力します。  
なお、ブレークチェックは行いません。

**戻 り 値** 0、あるいは入力した文字コードを返します。  
文字を出力した場合には必ず 0 を返します。  
入力時に戻り値が 0 ならば、入力が無かったことを示します。

**参 照 関 数** GETCHAR, PUTCHAR, GETC, INKEY

# INTVCG

レベル 1

**書式** #include <doslib.h>

```
int INTVCG (intno) ;  
int intno; /* 割り込みベクタ番号 */
```

**機能** intno で指定する割り込みベクタの処理アドレスを調べます。  
なお、割り込みベクタ 0x0~0xFF は割り込み処理、割り込みベクタ 0x100~0x1FF は IOCS コール、割り込みベクタ 0xFF00~0xFFFF は OS のファンクションコールに割りあてられています。

**戻り値** 割り込みベクタの処理アドレスを返します。

**参照関数** INTVCS

## プログラム例

```
#include <stdio.h>  
#include <doslib.h>  
  
#define IOCS 0x0100  
#define FUNC 0xff00  
  
main()  
{  
    int intno; /* ベクタ番号 */  
    intno = IOCS; /* IOCS */  
    printf("0x0100: 0x%x\n", INTVCG(intno));  
    intno = FUNC; /* ファンクションコール */  
    printf("0xff00: 0x%x\n", INTVCG(intno));  
}
```

# INTVCS

## レベル 1

**書式** #include <doslib.h>

```
int INTVCS (intno, jobadr) ;
int intno; /* ベクタの番号 */
char *jobadr; /* 処理ルーチンのエントリアドレス */
```

**機能** intno で指定する割り込みベクタに、jobadr で示す処理ルーチンの処理アドレスを設定します。

intno に指定できるベクタは次の通りです。

```
0x00~0xFF : 割り込み処理
0x100~0x1FF : IOCS コール
0xFF00~0xFFFF : OS のファンクションコール
```

**戻り値** 設定変更前のベクタの処理アドレスを返します。

**参照関数** INTVCG

### プログラム例

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

#define INR 0x6c
#define INRADR 0xa000

main()
{
    int intno; /* ベクタ番号 */
    char *jobadr; /* 処理ルーチンエントリ */

    intno = INR;
    jobadr = (char *)INRADR;

    INTVCS(intno, jobadr); /* ベクタ設定 */

    printf("0x%x: 0x%x\n", intno, INTVCG(intno));
}
```

# IOCTRLDVCTL

レベル 1

**書 式** # include <doslib.h>

```
int IOCTRLDVCTL (drive, f_code, buffer);
int drive; /* ドライブ番号 */
int f_code; /* ファンクション番号 */
unsigned char * buffer; /* 情報格納領域へのポインタ */
```

**機 能**

ドライブ番号 drive とファンクション番号 f\_code を指定して、デバイスドライバによる特殊コントロールを行います。

buffer には、デバイスドライバとの情報の受け渡しを行うための領域を指定します。

必要となる領域の大きさはデバイスドライバにより異なります。

処理内容はデバイスドライバにより異なります。

**戻 り 値**

戻り値は、デバイスドライバにより異なります。

負の数ならエラーを表します。

**参 照 関 数**

IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

# IOCTRLDVGT

## レベル 1

**書 式** # include <doslib.h>

```
int IOCTRLDVGT (drive) ;  
int drive; /* 調べたいドライブ番号 */
```

**機 能** 指定されたドライブがローカルかリモートかを調べます。

**戻 り 値** 指定されたドライブがローカルならば、デバイスドライバのデバイス情報を返します。  
リモートならば、戻り値のビット 12 が 1 になります。  
負の数ならエラーを表します。

**参 照 関 数** IOCTRLDVCTL, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

参 照 関 数

# IOCTRLFDCTL

レベル 1

**書 式** # include <doslib.h>

```
int IOCTRLFDCTL (fineno, f_code, buffer) ;
int fileno; /* ファイルハンドル */
int f_code; /* ファンクション番号 */
unsigned char * buffer; /* 情報格納領域へのポインタ */
```

**機 能** ファイルハンドル fileno とファンクションコード f\_code を指定して、デバイスドライバによる特殊コントロールを行います。

buffer は、デバイスドライバとの情報の受け渡しを行うための領域を指定します。

必要となる領域の大きさはデバイスドライバにより異なります。

処理内容はデバイスドライバにより異なります。

**戻 り 値** 戻り値は、デバイスドライバにより異なります。  
負の数ならエラーを表します。

**参 照 関 数** IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

# IOCTRLFDGT

レベル 1

**書 式** # include <doslib.h>

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

**機 能** 指定されたファイルがローカルかリモートかを返します。

**戻 り 値** 指定されたファイルがローカルならば、デバイスドライバのデバイス情報を返します。  
リモートならば、戻り値のビット 12 が 1 になります。  
負の数ならエラーを表します。

**参 照 関 数** IOCTLRDVCTL, IOCTLRDVGT, IOCTLFDCTL, IOCTLGT, IOCTRLIS, IOCTRLLOS, IOCTLRD, IOCTLRH, IOCTLRTSET, IOCTLST, IOCTLWD, IOCTLWH

# IOCTRLGT

レベル 1

**書 式** #include <doslib. h>

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

**機 能** fileno で指定されたファイルハンドルのデバイス情報を調べます。  
本関数は、デバイスドライバより直接入出力を行うときに使用します。

**戻 り 値** デバイス情報を返します。  
デバイス情報の各ビットが1であれば次の内容を表示します。

- ビット 0 標準入力デバイス (CON) であることを表す
- ビット 1 標準出力デバイス (CON) であることを表す
- ビット 2 NUL デバイスであることを表す
- ビット 3 CLOCK デバイスであることを表す
- ビット 5 RAW モードであることを表す (0 ならば COOKED モード)
- ビット 6 特殊 IOCTRL が可能であることを表す  
このビットが1でないと、IOCTRLDVCTL, IOCTRLFDCTL は  
使用できない
- ビット 7 キャラクタデバイスであることを表す (0 ならばブロックデバイス  
であり、ビット 0~4 はドライブ番号を表す)
- ビット 12 特殊ブロックデバイスの時、デバイスがリモートであることを表  
す (0 ならばローカル)  
ただし、ビット 13 が1の時のみ意味を持つ
- ビット 13 特殊ブロックデバイスであることを表す (0 ならば通常のブロック  
デバイス)  
ただし、ビット 7 が0の時のみ意味を持つ
- ビット 14 IOCTRL が可能であることを表す  
このビットが1でないと、IOCTRLRH, IOCTRLWH, IOCTR-  
LRD, IOCTRLWD は使用できない

その他のビットはシステムで予約

ビット 7 が0の場合 (ブロックデバイスの場合)  
ビット 0~4 の5ビットはドライブ番号 (0~25, A: = 0) を表します。  
負の数ならエラー表します。

## 参照関数

IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLIS, IOCTRLLOS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

## プログラム例

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

main()
{
    int      fno;          /* ファイルハンドル */

    fno = open("A:¥¥AUTOEXEC.BAT", O_RDWR);
    printf("0x%x¥n", IOCTRLGT(fno));
    close(fno);

    fno = open("CON", O_RDWR);
    printf("0x%x¥n", IOCTRLGT(fno));
    close(fno);
}
```

# IOCTRLIS

レベル 1

**書式** #include <doslib.h>

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

**機能** 指定したファイルハンドルの入力ステータスを調べます。

**戻り値** 次のいずれかのステータスを返します。

-1 : 入力可  
0 : 入力不可

**参照関数** IOCTLDVCTL, IOCTLDVGT, IOCTLFDCTL, IOCTLFDGT, IOCTLGT, IOCTRLS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH

## プログラム例

```
#include <stdio.h>  
#include <fcntl.h>  
#include <stat.h>  
#include <io.h>  
#include <doslib.h>  
  
main()  
{  
    int fno; /* ファイルハンドル */  
    int status; /* 入力ステータス */  
  
    fno = open("A:¥¥CONFIG.SYS", O_RDONLY);  
    status = IOCTLIS(fno);  
    printf("0x%02x¥n", status);  
}
```

# IOCTRLOS

レベル 1

**書式** #include <doslib.h>

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

**機能** 指定したファイルハンドルの出力ステータスを調べます。**戻り値** 次のいずれかのステータスを返します。

```
-1 : 出力可
0  : 出力不可
```

**参照関数** IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT, IOCTRLGT, IOCTRLIS, IOCTRLRD, IOCTRLRH, IOCTRLRTSET, IOCTRLST, IOCTRLWD, IOCTRLWH**プログラム例**

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

main()
{
    int fno; /* ファイルハンドル */
    int status; /* 出力ステータス */

    fno = open("A:¥¥CONFIG.SYS", O_RDONLY);
    status = IOCTRLOS(fno);
    printf("0x%02x¥n", status);
}
```

# IOCTRLRD

レベル 1

**書 式** #include <doslib.h>

```
int IOCTRLRD (drive, buffer, size) ;
int drive; /* ドライブ番号 */
unsigned char *buffer; /* 読み込みデータ格納領域へのポインタ */
int size; /* 読み込みバイト数 */
```

**機 能** driveで指定するドライブのデバイスドライバから、sizeで指定するバイト数を、bufferが指す領域へ読み込みます。

IOCTRLRDは、デバイスドライバより直接入力を行うときに使用します。

**戻 り 値** 正常終了の場合、読み込んだバイト数を返します。  
負の数ならばエラーを表します。

**参 照 関 数** IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT,  
IOCTRLGT, IOCTRLIS, IOCTRLIS, IOCTRLRH, IOCTRLRTSET,  
IOCTRLST, IOCTRLWD, IOCTRLWH

## プログラム例

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

#define DRIVE 0 /* ドライブ番号 */
#define SIZE 256 /* データbyte数 */

main()
{
    char buf[SIZE]; /* データ格納領域 */
    int n; /* 読んだデータ数 */

    if ((n = IOCTRLRD(DRIVE, buf, SIZE)) >= 0)
        printf("%d byte 読みました。%n", n);
    else {
        if ((IOCTRLDVGT(DRIVE) & 0x4000) == 0)
            printf("IOCTRLRDできないデバイス%n");
        else
            printf("エラーです。%n");
    }
}
```

# IOCTRLRH

レベル 1

**書式** #include <doslib.h>

```
int IOCTRLRH (fileno, buffer, size) ;
int fileno; /* ファイルのファイルハンドル */
unsigned char *buffer; /* 読み込みデータ格納領域へのポインタ */
int size; /* 読み込みバイト数 */
```

**機能** fileno で指定するファイルハンドルのデバイスドライバから、size で指定するバイト数を、buffer が指す領域へ読み込みます。  
IOCTRLRH 関数は、デバイスドライバより直接入力を行うときに使用します。

**戻り値** 正常終了の場合、読み込んだバイト数を返します。  
負の数ならエラー表します。

**参照関数** IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT,  
IOCTRLGT, IOCTRLIS, IOCTRLLOS, IOCTRLRD, IOCTRLRTSET,  
IOCTRLST, IOCTRLWD, IOCTRLWH

**プログラム例**

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

#define SIZE 128 /* データbyte数 */

main()
{
    char buf[SIZE]; /* データ格納領域 */
    int fno; /* ファイルハンドル */
    int n; /* 読んだデータ数 */

    fno = open("A:%#CONFIG.SYS", O_RDONLY);

    if ((n = IOCTRLRH(fno, buf, SIZE)) >= 0)
        printf("%d byte 読みました。%n", n);
    else {
        if ((IOCTRLFDGT(fno) & 0x4000) == 0)
            printf("IOCTRLRHできないデバイス%n");
        else
            printf("エラーです。%n");
    }
    close(fno);
}
```

# IOCTRLTSET

レベル 1

**書 式** # include <doslib. h>

```
int IOCTRLTSET (count, time);
int count; /* リトライ回数 */
int time; /* 時間 */
```

**機 能** リトライ数を設定します。

デフォルトの値は3回で、時間は100 (1秒) です。  
この関数はデバイスドライバでサポートされます。

**戻 り 値** 負の数ならエラーを表します。

**参 照 関 数** IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT,  
IOCTRLGT, IOCTRLLOS, IOCTRLRD, IOCTRLRH, IOCTRLST,  
IOCTRLWD, IOCTRLWH

```

#include <stdio.h>
#include <fcntl.h>
#include <sys.h>
#include <i.o.h>
#include <doslib.h>

#define FILE_SIZE 128

main()
{
    char buf[FILE_SIZE];
    int fd;
    int n;

    fd = open("A:\VOCORHIO.SYS", O_WRONLY);
    if (fd == IOCTRLRH(fno, buf, FILE_SIZE) == 0)
        printf("fd open 成功しました。fd: %d\n", fd);
    else
        printf("エラーです。fd: %d\n", fd);

    close(fd);
}

```

# IOCTRLST

## レベル 1

**書 式** #include <doslib.h>

```
int IOCTLST (fileno, data) ;
int fileno; /* ファイルのファイルハンドル */
int data; /* 設定するデバイス情報 */
```

**機 能** fileno で指定されたファイルハンドルにデバイス情報を設定します。

fileno は、キャラクタデバイスに対してオープンされたファイルハンドルでなければなりません。

設定できるのは、ビット 5 の RAW モード設定のみで、次の意味をもちます。

0x20 : RAW モード指定

0x00 : COOKED モード指定

**戻 り 値** 変更後のデバイス情報を返します。

負の数ならエラーを表します。

**参 照 関 数** IOCTLRDVCTL, IOCTLRDVGT, IOCTLRFCTL, IOCTLRFDT,  
IOCTRLGT, IOCTRLIS, IOCTRLIS, IOCTRLRD, IOCTRLRH,  
IOCTRLRTSET, IOCTLRLWD, IOCTLRLWH

### プログラム例

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

#define RAW 0x20
#define COOKED 0x00

main()
{
    int fno; /* ファイルハンドル */

    fno = open("CON", O_RDONLY);
    printf("old No. = %04x\n", IOCTLGT(fno));

    IOCTLST(fno, RAW);
    printf("new No. = %04x\n", IOCTLGT(fno));

    IOCTLST(fno, COOKED);
    printf("new No. = %04x\n", IOCTLGT(fno));

    close(fno);
}
```

# IOCTRLWD

レベル 1

**書式** #include <doslib.h>

```
int IOCTRLWD (drive, buffer, size) ;
int drive; /* ドライブ番号 */
unsigned char *buffer; /* 書き出しデータ格納領域へのポインタ */
int size; /* 書き出しバイト数 */
```

**機能** buffer が指す領域から、size で指定するバイト数のデータを、drive で指定するドライブのデバイスドライバに書き出します。

IOCTRLWD 関数は、デバイスドライバに直接出力を行うときに使用します。

**戻り値** 正常終了の場合、書き込んだバイト数を返します。  
負の数ならエラーを表します。

**参照関数** IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT,  
IOCTRLGT, IOCTRLIS, IOCTRLLOS, IOCTRLRD, IOCTRLRH,  
IOCTRLRTSET, IOCTRLST, IOCTRLWH

## プログラム例

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

#define DRIVE 0 /* ドライブ番号 */
#define SIZE 256 /* データbyte数 */

main()
{
    char buf[SIZE]; /* データ格納領域 */
    int n; /* 書いたデータ数 */

    if ((n = IOCTRLWD(DRIVE, buf, SIZE)) >= 0)
        printf("%d byte 書きました。%n", n);
    else {
        if ((IOCTRLDVGT(DRIVE) & 0x4000) == 0)
            printf("IOCTRLWD できないデバイス%zn");
        else
            printf("エラーです。%zn");
    }
}
```

## IOCTRLWH

レベル 1

**書式** #include <doslib.h>

```
int IOCTLWH (fileno, buffer, size) ;
int fileno; /* ファイルのファイルハンドル */
unsigned char *buffer; /* 書き出しデータ格納領域へのポインタ */
int size; /* 書き出しバイト数 */
```

**機能**

buffer が指す領域から、size で指定するバイト数を fileno で指定するファイルハンドルのデバイスドライバに書き出します。

IOCTRLWH 関数は、デバイスドライバに直接出力を行うときに使用します。

**戻り値**

正常終了の場合、書き込んだバイト数を返します。  
負の数ならエラーを表します。

**参照関数**

IOCTRLDVCTL, IOCTRLDVGT, IOCTRLFDCTL, IOCTRLFDGT,  
IOCTRLGT, IOCTRLIS, IOCTRLLOS, IOCTRLRD, IOCTRLRH,  
IOCTRLRTSET, IOCTRLST, IOCTRLWD

**プログラム例**

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

#define SIZE 256 /* データbyte数 */

main()
{
    char buf[SIZE]; /* データ格納領域 */
    int fno; /* ファイルハンドル */
    int n; /* 書いたデータ数 */

    fno = open("A:¥¥CONFIG.SYS", O_RDWR);

    if ((n = IOCTLWH(fno, buf, SIZE)) >= 0)
        printf("%d byte 書きました。¥n", n);
    else {
        if ((IOCTRLFDGT(fno) & 0x4000) == 0)
            printf("IOCTRLWHできないデバイス¥n");
        else
            printf("エラーです。¥n");
    }
    close(fno);
}
```



# ISNS232C

レベル 0

**書 式** #include <iocslib.h>

```
int ISNS232C ( );
```

**機 能** RS-232C 受信データをチェックします。

**戻 り 値** 0x10000+受信データを返します。

戻り値が0のときは、受信データがないことを示します。

## プログラム例

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

main()
{
    int data; /* 受信データ */

    data = ISNS232C();
    if (data == 0)
        puts("受信データがありません。");
}
```

**書 式** #include <iocslib.h>

```
int JISSFT (CODE) ;  
int CODE; /* JIS 漢字コード */
```

**機 能** JIS 漢字コードをシフト JIS コードに変換します。

**戻 り 値** シフト JIS コードを返します。  
ただし、上位 16 ビットが 0xFFFF ならエラーが発生したことを示します。

### プログラム例

```
#include <stdio.h>  
#include <iocslib.h>  
  
main()  
{  
    int jis; /* JIS 漢字コード */  
    int ms; /* シフト JIS コード */  
  
    jis = 0x4f53;  
    ms = JISSFT(jis);  
  
    printf("0x%04x(jis) : 0x%04x(sft)%n", jis, ms);  
}
```

# JOYGET

レベル 0

**書 式** #include <iocslib.h>

```
int JOYGET (STICKNO) ;
int STICKNO /* ジョイスティック番号 */
```

**機 能**

ジョイスティックのデータを読み込みます。

ビットが0ならばONであることを示します。

STICKNO にはジョイスティック番号を指定します (0 ならばジョイスティック 1、1 ならばジョイスティック 2 となります)。

**戻 り 値**

次の値を返します。

1	S2	S1	1	R	L	D	U
---	----	----	---	---	---	---	---

B ボタン A ボタン

右 左 下 上

# KEEPPR

レベル 1

**書 式** #include <doslib.h>

```
void KEEPPR (prglen, code) ;  
int prglen; /* プロセスの常駐バイト数 */  
int code; /* 終了コード */
```

**機 能** プロセスの総バイト数のうち、prglenで指定されたバイト数（PSP 部分を除く）の部分を常駐させたまま、プロセスを終了させます。  
現在オープンされているファイルハンドルは、すべてクローズされます。  
子プロセスがオープンしたファイルハンドルについても同じです。  
prglenは `_HEND - _PSP - 0xf0` で求めることができます。  
`_HEND` と `_PSP` は int の外部変数です。

**戻 り 値** 戻り値はありません。

**参 照 関 数** EXIT, EXIT2

## プログラム例

```
#include <doslib.h>  
  
extern int _HEND;  
extern int _PSP;  
  
main()  
{  
    int code; /* 終了コード */  
  
    code = 0;  
  
    KEEPPR(_HEND - _PSP - 0xf0, code);  
}
```

# KEYSNS

## レベル 1

**書 式** #include <doslib.h>

```
int KEYSNS ( );
```

**機 能** 標準入力の入力状態を調べます。  
また、このときブレークチェックも行います (^C : ^P : ^N: )。

**戻 り 値** 入力があるときには-1、ないときは0を返します。

# KFLUSHGC

レベル 1

**書 式** #include <doslib.h>

int KFLUSHGC ( ) ;

**機 能**

入力バッファをフラッシュしてから標準入力から1文字読み込みます。

このときブレークチェックも行います (^C:^P:^N:).

KFLUSHGC 関数は入力バッファをフラッシュして GETC 関数と同様な処理を行います。

**戻 り 値**

読み込んだ文字コードを返します。

**参 照 関 数**

GETC, KFLUSHGP, KFLUSHIO, KFLUSHIN, KFLUSHGS

# KFLUSHGP

## レベル 1

**書式** #include <doslib.h>

```
int KFLUSHGP ( ) ;
```

**機能** 入力バッファをフラッシュしてから標準入力から1文字読み込み、読み込んだ文字を標準出力にエコーバックします。  
このときブレークチェックも行います (^C : ^P : ^N :)。  
KFLUSHGP 関数は入力バッファをフラッシュして GETCHAR 関数と同様な処理を行います。

**戻り値** 読み込んだ文字コードを返します。

**参照関数** GETCHAR, KFLUSHIO, KFLUSHIN, KFLUSHGC, KFLUSHGS

# KFLUSHGS

レベル 1

**書 式** #include <doslib.h>

```
int KFLUSHGS (inpptr) ;
struct INPPTR { /* 入力文字列格納領域 */
    unsigned char max ; /* 入力時の最大文字数 */
    unsigned char length ; /* 入力が行われた文字数 */
    unsigned char buffer [256] ; /* 入力データ */
} * inpptr ;
```

**機 能** 入力バッファをフラッシュしてから CR コードが入力データに現れるまで、文字列を入力します。

このときブレークチェックも行います (^C : ^P : ^N :)。

入力が最大文字数 `inpmax` を超えた場合は、警告音を出しますが終了はしません。

`length` には実際に入力された文字数が、`buffer` には入力された文字列データが格納されます。

KFLUSHGS 関数は入力バッファをフラッシュして GETS 関数と同様な処理を行います。

**戻り値** CR コードを除く入力文字数を返します。

**参照関数** GETS, KFLUSHGP, KFLUSHIO, KFLUSHIN, KFLUSHGC

# KFLUSHIN

レベル 1

**書 式** #include <doslib.h>

```
int KFLUSHIN ( ) ;
```

**機 能** キーバッファをフラッシュしてからキーが押下されるまで待ち、文字コードを入力します。

ブレイクチェックは行いません。

KFLUSHIN 関数はキーバッファをフラッシュして、INKEY 関数と同様な処理を行います。

**戻 り 値** 入力された文字コードを返します。

**参 照 関 数** INKEY, KFLUSHGCP, KFLUSHGIO, KFLUSHGHC, KFLUSHGHS

# KFLUSHIO

レベル 1

**書 式** #include <doslib. h>

```
int KFLUSHIO (code) ;  
int code; /* ファンクションコード */
```

**機 能**

キーバッファをフラッシュしてから、コンソールの直接入出力を行います。  
code が 0xFF ならばキー入力を行い、入力がない場合は待つことはしません。  
code が 0xFE ならばキーのセンスを行い、入力した文字コードを返します。  
これ以外のときは、code を文字コードとして出力します。  
なお、ブレイクチェックは行いません。  
KFLUSHIO 関数はキーバッファをフラッシュして INPOUT 関数と同様な処理を行います。

**戻り値**

0、あるいは入力した文字コードを返します。  
文字を出力した場合には必ず 0 を返します。  
入力時に戻り値が 0 ならば、入力があったことを示します。

**参照関数**

INPOUT, KFLUSHGP, KFLUSHIN, KFLUSHGC, KFLUSHGS

# KILL\_PR

レベル 1

**書式** # include <doslib. h>

```
int KILL_PR ( );
```

**機能** 自分自身のプロセスを削除します。

常駐終了していたプロセスの場合、同じ psp\_id を持つスレッドを全て削除し、確保されていたメモリを開放します。

KILL\_PR 関数を実行する時は、自分でオープンしたファイルは全てクローズし、書き換えたベクタなども元に戻した後にしてください。

システムは、スレッドの削除とメモリの開放しか行いません。

プロセス内でいくつかのスレッドを登録し実行した後で、EXIT 関数や EXIT2 関数でプロセスを終了する場合は、プロセス内でオープンしたスレッドのみを KILL\_PR 関数で削除してから、プロセスを終了してください。

メインのスレッドであるプロセスを KILL\_PR 関数した場合、その後の動作は保証されません。

**戻り値** 負の数ならエラーを表します。

**参照関数** OPEN\_PR, GET\_PR, SUSPEND\_PR, SEND\_PR, TIME\_PR, CHANGE\_PR

# K\_INSMOD

レベル 1

**書 式** #include <doslib.h>

```
void K_INSMOD (n) ;  
int n; /* INS キーのモード */
```

**機 能**

INS キーのモードを切り替えます。

n に 0xFF を指定したときには ON、0x00 を指定したときには OFF となります。

**戻 り 値**

戻り値はありません。

**参 照 関 数**

K\_KEYINP, K\_KEYSNS, K\_SFTSNS, K\_KEYBIT

**プログラム例**

```
#include <stdio.h>  
#include <doslib.h>  
  
#define INS_ON 0xff  
#define INS_OFF 0x00  
  
main()  
{  
    K_INSMOD(INS_ON); /* 挿入モード */  
    puts("挿入モードです。");  
  
    puts("Hit any key");  
    while (K_KEYSNS() == 0)  
        ;  
  
    K_INSMOD(INS_OFF); /* 非挿入モード */  
    puts("非挿入モードです。");  
}
```

# K\_KEYBIT

## レベル 1

**書式** #include <doslib.h> ,

```
int K_KEYBIT (n) ;
int n; /* キーコードグループ */
```

**機能** 次に示すキーコードグループを指定して、キーの押下状態を調べます。

キーコードグループ \ ビット	0	1	2	3	4	5	6	7
0	未定義	ESC	1 !	2 "	3 #	4 \$	5 %	6 &
1	7 '	8 (	9 )	0	- =	^	¥	BS
2	TAB	Q	W	E	R	T	Y	U
3	I	O	P	@	[	CR	A	S
4	D	F	G	H	J	K	L	;
5	:	]	Z	X	C	V	B	N
6	M	, <	. >	/ ?	_	SP	HOME	DEL
7	Roll up	Roll down	UNDO	←	↑	→	↓	CLR
8	/	*	-	7	8	9	+	4
9	5	6	=	1	2	3	ENTER	0
A	,	.	記号	登録	HELP	XF1	XF2	XF3
B	XF4	XF5	かな	ローマ字	コード	CAPS	INS	ひらがな
C	全角	BREAK	COPY	F・1	F・2	F・3	F・4	F・5
D	F・6	F・7	F・8	F・9	F・10	未定義	未定義	未定義
E	SHIFT	CTRL	OPT.1	OPT.2	未定義	未定義	未定義	未定義
F	未定義	未定義	未定義	未定義	未定義	未定義	未定義	未定義

このファンクションコールは、CON デバイスによりサポートされます。

**戻り値** 上記の表に示す各ビットが0か1かにより、各キーコードグループのキーの押下状態を返します。

ビットが1になっていれば、キーが押されている状態です。

**参照関数** K\_KEYINP, K\_KEYSNS, K\_SFTSNS, K\_INSMOD



# K\_KEYINP

レベル 1

**書 式** #include <doslib.h>

```
int K_KEYINP ( ) ;
```

**機 能** キーボードからの入力を行います。  
このファンクションコールは、CON デバイスによりサポートされます。

**戻 り 値** 入力した文字コードを返します。

**参 照 関 数** K\_KEYSNS, K\_SFTSNS, K\_KEYBIT, K\_INSMOD

# K\_KEYSNS

レベル 1

**書 式** #include <doslib.h>

```
int K_KEYSNS ( ) ;
```

**機 能** キーボードからの入力を読み取ります。  
ただし、'0'のときは入力が行われなかったことを示します。  
このファンクションコールは、CON デバイスによりサポートされます。

**戻 り 値** 読み取った文字コードを返します。

**参 照 関 数** K\_KEYINP, K\_SFTSNS, K\_KEYBIT, K\_INSMOD

# K\_SFTSNS

レベル 1

**書式** #include <doslib.h>

```
int K_SFTSNS ( );
```

**機能** シフトキーの押下状態を調べます。  
なお、このファンクションコールは、CON デバイスによりサポートされます。

**戻り値** シフトキーの押下状態を返します。  
ただし、LED の付いているキーは、LED の点灯状態を返します。  
ビット 10~ビット 0 の内容は次の通りです。

10	9	8	7	6	5	4	3	2	1	0
全角	ひらが な	INS	CAPS	コード	ローマ	かな	opt2	opt1	ctrl	shift

(いずれもビットが1のとき、押下状態であることを示します)。

**参照関数** K\_KEYINP, K\_KEYSNS, K\_KEYBIT, K\_INSMOD

## プログラム例

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

main()
{
    int shtstas; /* シフトキー状態 */

    shtstas = K_SFTSNS();
    printf("シフトキー: %11b\n", shtstas);
}
```

# LEDMOD

レベル 0

**書 式** #include <ioclib.h>

```
void LEDMOD (KEYCODE, ONOFF) ;  
int KEYCODE; /* キー番号 */  
int ONOFF; /* ON/OFF モード */
```

**機 能** LED キーのモードを設定します。

ON/OFF には 1 (ON)、または 0 (OFF) を指定します。

KEYCODE には 0~6 を指定します。

その内容は次の通りです。

6	5	4	3	2	1	0
全角	ひらがな	INS	CAPS	コード入力	ローマ字	かな

**戻り値** 戻り値はありません。

```
#include <ioclib.h>  
#include <ioclib.h>  
  
main()  
{  
    int shatas;  
    shatas = K_STNS;  
    printf("シャイキー: %i\n", shatas);  
}
```

# LINE

レベル 0

**書 式** #include <ioclib.h>

```
int LINE (lineptr);
struct LINEPTR {
    short x1; /* 始点の X 座標 */
    short y1; /* 始点の Y 座標 */
    short x2; /* 終点の X 座標 */
    short y2; /* 終点の Y 座標 */
    unsigned short color; /* パレットコード */
    unsigned short linestyle; /* ラインスタイル */
} *lineptr;
```

**機 能** グラフィック画面にラインを引きます。

**戻 り 値** 終了コードを返します。

- 0 : 正常終了
- 1 : グラフィックは使用不可

# LOAD

レベル 1

**書 式** #include <doslib. h>

```
int LOAD (file, comline, envptr) ;
unsigned char *file; /* プログラムファイル名へのポインタ */
struct COMLINE {
    unsigned char len; /* コマンドラインの長さ */
    unsigned char buffer [255]; /* コマンドライン文字列 */
} *comline;
unsigned char *envptr; /* 環境文字列へのポインタ */
```

**機 能**

comline で指定するコマンドラインと、envptr で指定する環境にもとづいて、file で指定するプログラムをロードします。  
envptr に 0 を指定すると、呼び出し側のプログラムと同じ環境になります。  
file の最上位 8 ビットの意味は次の通りです。

- 0 : 拡張子 (. R/. Z/. X) にしたがってロードする
- 1 : R タイプのファイルとしてロードする
- 2 : Z タイプのファイルとしてロードする
- 3 : X タイプのファイルとしてロードする

**戻 り 値**

ロードしたプログラムの実行アドレスを返します。  
負の数ならエラーを表します。

**参 照 関 数**

LOADEXEC, PATHCHK, LOADONLY, EXECONLY

# LOADEXEC

レベル 1

**書 式** #include <doslib.h>

```
int LOADEXEC (file, comline, envptr) ;
/* unsigned char *file; /* プログラムファイル名へのポインタ */
struct COMLINE {
    unsigned char len; /* コマンドラインの長さ */
    unsigned char buffer [255]; /* コマンドライン文字列 */
} *comline;
unsigned char *envptr; /* 環境文字列へのポインタ */
```

**機 能** comline で指定するコマンドラインと、envptr で指定する環境にもとづいて、file で指定するプログラムをロード、実行します。  
envptr に 0 を指定すると、呼び出し側のプログラムと同じ環境になります。  
file の最上位 8 ビットの意味は次の通りです。

- 0 : 拡張子 (. R/. Z/. X) にしたがってロードする
- 1 : R タイプのファイルとしてロードする
- 2 : Z タイプのファイルとしてロードする
- 3 : X タイプのファイルとしてロードする

**戻り値** ロード、実行ができなかった場合にはエラーコード (負の数) を、正常終了した場合にはプロセス終了コード (正の数) を返します。

**参照関数** LOAD, PATHCHK, LOADONLY, EXEONLY

# LOADONLY

レベル 1

**書 式** #include <doslib.h>

```
int LOADONLY (file, loadadr, limitadr) ;
/* unsigned char *file; /* ファイル名格納領域へのポインタ */
/* unsigned char *loadadr; /* ロードアドレス */
/* unsigned char *limitadr; /* リミットアドレス */
```

**機 能** loadadr, limitadr にそれぞれロードアドレス、リミットアドレスを指定して、file のプログラムをロードします。file の最上位 8 ビットの意味は次の通りです。

- 0 : 拡張子 (.R/.Z/.X) にしたがってロードする
- 1 : R タイプのファイルとしてロードする
- 2 : Z タイプのファイルとしてロードする
- 3 : X タイプのファイルとしてロードする

**戻り値** プログラムのサイズを返します。  
負の数ならエラーを表します。

**参照関数** LOADEXEC, LOAD, PATHCHK, EXECONLY

# LOCK

## レベル 1

**書式** # include <doslib.h>

```
int LOCK (fileno, offset, len) ;
int fileno; /* ファイルハンドル */
int offset; /* ロックする部分へのオフセット */
int len; /* ロックする部分の長さ */
```

**機能**

ファイルのロックを行い、他のプロセスからのファイルアクセスを禁止します。fileno は、ロックしたいファイルのファイルハンドルを指定します。offset には、ファイル中のロックしたい部分の、ファイルの先頭からのオフセットを指定します。len には、ファイル中のロックしたい部分のバイト数を指定します。この LOCK 関数と UNLOCK 関数により、ファイルアクセスの排他制御を行い、自分以外のプロセスが勝手にファイルを書き変えるのを防ぐことができます。

**戻り値**

負の数ならエラーを表します。

**参照関数**

UNLOCK

**プログラム例**

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

#define SIZE 256

static char buf[SIZE];

main()
{
    int fd;

    if ((fd = open("A:¥¥vjeb.dic", O_RDONLY|O_BINARY)) == -1)
        puts("can't open");
    else if (LOCK(fd, 0, SIZE) < 0)
        puts("can't lock");
    else if (read(fd, buf, SIZE) != SIZE)
        puts("can't read");
    if (UNLOCK(fd, 0, SIZE) < 0)
        puts("can't unlock");
    close(fd);
}
```

# LOF232C

レベル 0

**書式** #include <iocslib.h>

int LOF232C ( );

**機能** RS-232C 受信バッファ内のデータ数を調べます。

**戻り値** RS-232C 受信バッファ内のデータ数を返します。

## プログラム例

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

main()
{
    int n; /* データ数 */
    n = LOF232C();
    printf("data in buffer = %dbytes\n", n);
}
```

# MAKEASSIGN

レベル 1

**書式** # include <doslib. h>

```
int MAKEASSIGN (buff1, buff2, mode) ;  
unsigned char *buff1; /* ドライブ名 */  
unsigned char *buff2; /* ディレクトリ名 */  
int mode; /* 割り当てモード */
```

**機能** 仮想ドライブ、仮想ディレクトリの割り当てを行います。  
mode の値により、以下のような割り当てを行います。

- mode が 0x50 のとき (仮想ドライブの割り当て)  
buff1 のドライブをアクセスすると、buff2 のディレクトリをアクセスします。
- mode が 0x60 のとき (仮想ディレクトリの割り当て)  
buff2 のディレクトリをアクセスすると、buff1 のドライブをアクセスします。

**戻り値** 負の数ならエラーを表します。

**参照関数** GETASSIGN, RASSIGN

# MAKETMP

レベル 1

**書 式** # include <doslib. h>

```
int MAKETMP (file, atr) ;  
unsigned char *file; /* パス名格納領域へのポインタ */  
int atr; /* ファイル属性 */
```

**機 能** 指定したパスにテンポラリファイルを作成します。

file には、テンポラリファイルを作成するディレクトリを示すパス名を設定し、続けて次に示すようなファイル名を設定します。

file????.txt

????の部分は、0000~9999までの数字に変換されます。

????の代わりに数字を設定すると、その数から検索され、ファイル名が決定します。

atr には、作成するテンポラリファイルの属性を指定します。

ファイル属性の指定方法は CREATE 関数と同じなので、そちらを参照してください。

**戻 り 値** 作成したテンポラリファイルのファイルハンドルを返します。

負の数ならエラーを表します。

**参 照 関 数** CREATE

## プログラム例

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

static char *template = "stm0000";
static char tmp[256];

main()
{
    int fd1;
    int fd2;

    strcpy(tmp, template);
    if ((fd1 = MAKETMP(tmp, 0x20)) < 0)
        fprintf(stderr, "can't open temp file\n");
    strcpy(tmp, template);
    if ((fd2 = MAKETMP(tmp, 0x20)) < 0)
        fprintf(stderr, "can't open temp file\n");

    WRITE(fd1, "hello", 6);
    WRITE(fd2, "hello", 6);

    CLOSE(fd2);
    CLOSE(fd1);

    system("dir");
    exit(0);
}
```

# MALLOC

## レベル 1

**書式** #include <doslib.h>

```
int MALLOC (size) ;
int size; /* 確保する領域のサイズ */
```

**機能** size で指定されたバイト数のメモリ領域を確保し、そのアドレスを返します。  
size に 0x1000000 以上の値を指定したときは、0xFFFFFFFF として処理します。

**戻り値** メモリが確保できたときは確保した領域を指すアドレスを返します。  
確保できなかったときには 0x81000000+最大バイト数を返します。  
0x82000000は、まったく確保できないことを示します。

# MALLOC2

## レベル 1

**書 式** # include <doslib. h>

```
int MALLOC2 (md, size) ;  
int md; /* メモリ割り当てのモード */  
int size; /* 確保したい領域のサイズ */
```

**機 能**

size で指定されたバイト数のメモリ領域を確保し、そのポインタを返します。  
md には、以下に示すメモリの割り当て方法を指定します。

- 0: 下位から探し、割り当てる。
- 1: 指定されたサイズを満たすメモリブロックのうち最小ブロックを割り当てる。
- 2: 上位の方から探し、割り当てる。

size に 0x1000000 以上の値を指定した時は、0xFFFFFFFF として処理します。

**戻 り 値**

メモリが確保できた時は、その領域を指すポインタを返します。  
確保できなかった時は、0x81000000+最大バイト数を返します。  
0x82000000?はまったく確保できないことを表します。

**参 照 関 数**

MALLOC, MFREE

# MFREE

レベル 1

**書 式** #include <doslib.h>

```
int MFREE (memptr) ;  
int memptr; /* メモリブロックを指すアドレス */
```

**機 能** memptr で指定したメモリブロックを解放します。

memptr は MALLOC が返した値です。  
したがって、誤った memptr を指定するとエラーになります。  
memptr に 0 を指定すると、確保したメモリ領域をすべて解放します。  
また、子プロセスが確保した領域についても同様です。

**戻 り 値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) でエラー。

# MKDIR

## レベル 1

**書式** #include <doslib.h>

```
int MKDIR (file) ;  
unsigned char *file; /* ディレクトリ名格納領域へのポインタ */
```

**機能** ディレクトリを作成します。

**戻り値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) 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は、0 以外の正の数の場合があります。

**参照関数** RMDIR, CHDIR

# MOVE

レベル 1

**書 式** #include <doslib.h>

```
int MOVE (old, new) ;
unsigned char *old; /* 旧ファイル名へのポインタ */
unsigned char *new; /* 新ファイル名へのポインタ */
```

**機 能** old で指定したファイルの名前を、new で示す名前に変更します。  
old と new のアクセスパスが異なる場合は、ディレクトリ間の移動も行います。  
ただし、ドライブが異なる場合には、移動はできません。  
変更の対象がディレクトリの場合、名前は変更はできますが、移動はできません。  
オープンされているファイルに対して実行した場合はエラーになります。

**戻 り 値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) でエラー。  
通常は 0 が返るが、特殊デバイスドライバが対象の時は、0 以外の正の数の場合がある。

**参 照 関 数** RENAME

## プログラム例

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

main()
{
    int      exist(char *);
    char     *old;
    char     *new;

    old = "A:¥¥CONFIG.SYS";
    new = "A:¥¥CONFIG.BAK";

    if (!exist(old))
        MOVE(new, old);          /* 変更(2回目) */
    else
        MOVE(old, new);         /* 変更(1回目) */
}

static int   exist(char *file)
{
    int      fd;

    if ((fd = open(file, O_RDONLY)) < 0)
        return (0);
    else {
        close(fd);
        return (1);
    }
}
```

# MS\_CURGT レベル 0

**書 式** #include <iocslib. h>

```
int MS_CURGT ( );
```

**機 能** マウスカーソルの座標を調べます。

**戻 り 値** マウスカーソルの座標を返します。  
 (上位 16 ビットに X 座標を、下位 16 ビットに Y 座標を設定して返します。)

```
XXXXXXXX XXXXXXXX YYYYYYYY YYYYYYYY
```

X 座標

Y 座標

# MS\_CUROF

レベル 0

**書 式** #include <iocslib. h>

void MS\_CUROF ( ) ;

**機 能** マウスカーソルを消します。

**戻 り 値** 戻り値はありません。

# MS\_CURON

レベル 0

**書 式** #include <iocslib. h>

void MS\_CURON ( ) ;

**機 能** マウスカーソルの表示を行います。

**戻 り 値** 戻り値はありません。

# MS\_CURST

レベル 0

**書 式** #include <iocslib.h>

```
int MS_CURST (X, Y) ;  
int X; /* X座標 */  
int Y; /* Y座標 */
```

**機 能** マウスカーソルの座標を指定します。

**戻り値** 処理が正常に終了したら 0、エラーが発生した場合には -1 を返します。

# MS\_GETDT

レベル 0

**書 式** #include <iocslib.h>

```
int MS_GETDT ( );
```

**機 能** マウスの移動量、およびボタンの ON/OFF を調べます。

**戻り値** マウスの移動量、およびボタンの ON/OFF の状態を返します。

```
XXXXXXXX YYYYYYYY LLLLLLLL RRRRRRRR
```

ビット 31~24 XXXXXXXX : X 方向移動量

ビット 23~16 YYYYYYYY : Y 方向移動量

ビット 15~8 LLLLLLLL : 左ボタンの状態 (0xFF = on, 0x00 = off)

ビット 7~0 RRRRRRRR : 右ボタンの状態 (0xFF = on, 0x00 = off)

# MS\_INIT

レベル 0

**書式** #include <iocslib.h>

void MS\_INIT ( );

**機能** マウスの初期化を行います。

**戻り値** 戻り値はありません。

# MS\_LIMIT

レベル 0

**書 式** #include <iocslib.h>

```
int MS_LIMIT (XS, YS, XE, YE) ;  
int XS; /* 先頭 X 座標 */  
int YS; /* 先頭 Y 座標 */  
int XE; /* 最終 X 座標 */  
int YE; /* 最終 Y 座標 */
```

**機 能** マウスカーソルの移動範囲を指定します。

**戻 り 値** 処理が正常に終了したら 0、エラーが発生した場合には -1 を返します。

# MS\_OFFTM

レベル 0

**書 式** #include <ioclib.h>

```
int MS_OFFTM (MODE, MAXTIME);  
int MODE; /* マウスボタンの指定 */  
int MAXTIME; /* 待ち時間の最大値 */
```

**機 能** マウスボタンを離すまでの時間を調べます。  
MODE にはマウスボタンを指定します。  
0 のときは左ボタン、-1 のときは右ボタンとなります。  
MAXTIME には、待ち時間の最大値を指定します。  
0 を指定すると、いつまでも待ち続けます。

**戻り値** 待ち時間を返します。(1~65534)。  
ドラッグが行われたときには0、待ち時間の最大値を超えたときには65535を返します。

# MS\_ONTM

レベル 0

## 書式

```
#include <ioclib.h>
```

```
int MS_ONTM (MODE, MAXTIME) ;  
int MODE; /* マウスボタンの指定 */  
int MAXTIME; /* 待ち時間の最大値 */
```

## 機能

マウスボタンを押すまでの時間を調べます。  
MODE にはマウスボタンを指定します。  
0 のときは左ボタン、-1 のときは右ボタンとなります。  
MAXTIME には、待ち時間の最大値を指定します。  
0 を指定すると、いつまでも待ち続けます。

## 戻り値

待ち時間を返します (1~65534)。  
ドラッグが行われたときには 0、待ち時間の最大値を超えたときには 65535 を返します。

# MS\_PATST

レベル 0

**書 式** #include <iocslib.h>

```

void MS_PATST (CURSORNO, ADDRESS) ;
int CURSORNO; /* カーソル番号 */
struct PATST *ADDRESS; /* パターンデータ格納アドレス */
struct PATST {
    short OFFSET X;
    short OFFSET Y;
    short shadow [16] ;
    short pattern [16] ;
};

```

**機 能**

マウスカーソルのパターンを登録します。  
CURSORNO には、マウスカーソルの番号を指定します。  
ADDRESS には、パターンデータ格納領域のアドレスを指定します。

**戻 り 値**

戻り値はありません。

## プログラム例

```
#include <iocslib.h>

#define CURSOLNO 1 /* カーソル番号 */
#define TYPE 8
#define FONTSIZE 32
#define CODE 0x8199

main()
{
    struct FNTBUF *bufp;
    struct PATST pat; /* パターンデータ */
    char *p;
    int i;

    i = sizeof(struct FNTBUF) + FONTSIZE;
    bufp = (struct FNTBUF *)malloc(i);
    /* shit! */

    FNTGET(TYPE, CODE, bufp);
    pat.OFFSETX = 0;
    pat.OFFSETY = 0;
    p = bufp->buffer;
    for (i = 0; i < 16; i++) {
        pat.shadow[i] = 0;
        pat.pattern[i] = (*p++ & 0xff) << 8;
        pat.pattern[i] += (*p++ & 0xff) << 0;
    }
    MS_PATST(CURSOLNO, &pat);

    MS_SEL(CURSOLNO);
    free((char *)bufp);
}
```

# MS\_SEL

レベル 0

**書 式** #include <iocslib. h>

```
void MS_SEL (CURSORNO) ;  
int CURSORNO; /* カーソル番号 */
```

**機 能** マウスカーソルを選択します。

CURSORNO にはマウスカーソルの番号を指定します。

**戻 り 値** 戻り値はありません。

# MS\_SEL2

レベル 0

**書 式** #include <iocslib.h>

```
void MS_SEL2 (TBLADDRESS) ;  
unsigned char * TBLADDRESS; /* カーソル番号テーブルアドレス  
*/
```

**機 能** マウスカーソルをいくつか指定して、アニメーションを作成します。  
TBLADDRESS には、カーソル番号テーブルのアドレスを指定します。  
カーソル番号の終わりは、short の -1 です。

**戻り値** 戻り値はありません。

TATS\_3M

## プログラム例

```

#include      <iocslib.h>

#define TYPE      8
#define FONTSIZE  32
#define CODE1     0x217b
#define CODE2     0x217c

main()
{
    struct PATST  pat1;    /* パターンデータ格納アドレス */
    struct PATST  pat2;    /* パターンデータ格納アドレス */
    struct FNTBUF *buf1;   /* フォントデータ格納アドレス */
    struct FNTBUF *buf2;   /* フォントデータ格納アドレス */
    short  curno[10];      /* カーソル番号テーブル */
    unsigned char *p1;
    unsigned char *p2;
    int     i;

    i = sizeof(struct FNTBUF) + FONTSIZE;
    buf1 = (struct FNTBUF *)malloc(i);
    buf2 = (struct FNTBUF *)malloc(i);
    /* shit ! */

    /* フォントデータの読み込み */
    FNTGET(TYPE, CODE1, buf1);
    FNTGET(TYPE, CODE2, buf2);

    /* パターンデータの設定 */
    p1 = buf1->buffer;
    p2 = buf2->buffer;
    for (i = 0; i < 16; i++) {
        pat1.shadow[i] = pat2.shadow[i] = 0;
        pat1.pattern[i] = (*p1++ & 0xff) << 8;
        pat1.pattern[i] += (*p1++ & 0xff) << 0;
        pat2.pattern[i] = (*p2++ & 0xff) << 8;
        pat2.pattern[i] += (*p2++ & 0xff) << 0;
    }
    MS_PATST(1, &pat1);    /* マウスカーソルの定義 */
    MS_PATST(2, &pat2);

    curno[0] = 1;          /* カーソル番号テーブルの設定 */
    curno[1] = 2;
    curno[2] = -1;

    MS_SEL2(curno);       /* アニメーション指定 */

    free(buf1);
    free(buf2);
}

```



# NAMECK

## レベル 1

**書式** #include <doslib.h>

```
int NAMECK (file, buffer) ;
unsigned char *file; /* ファイル名格納領域へのポインタ */
struct NAMECKBUF *buffer; /* 解析結果格納領域へのポインタ */
```

**機能** ファイル名の解析を行い、その結果を buffer に格納します。  
buffer の形式は次の通りです。

```
struct NAMECKBUF {
    unsigned char drive [2]; /* ドライブ */
    unsigned char path [65]; /* パス名, ルート = '\', サブ = '\test\'
                               */
    unsigned char name [19]; /* ファイル名 */
    unsigned char ext [5]; /* 拡張子 */
};
```

**戻り値** 負の数ならエラーを表します。

0xFF ならばファイル指定なし、0x00 ならワイルド指定なし、これ以外の正の数のときにはワイルド指定ありとなります。

### プログラム例

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

main()
{
    struct NAMECKBUF inf; /* 解析情報 */
    NAMECK("A:\%CONFIG.SYS", &inf); /* ファイル名解析 */
    printf("drive = %c%c\n", inf.drive[0], inf.drive[1]);
    printf("path = %s\n", inf.path);
    printf("name = %s\n", inf.name);
    printf("ext = %s\n", inf.ext);
}
```

# NAMESTS

レベル 1

**書式** #include <doslib.h>

```
int NAMESTS (file, buffer) ;
/* unsigned char *file; /* ファイル名格納領域へのポインタ */
/* struct NAMESTBUF *buffer; /* 解析情報格納領域へのポインタ */
```

**機能** file が示すファイル名の解析を行い、解析結果を buffer へ格納します。  
buffer の形式は次の通りです。

```
struct NAMESTBUF {
    unsigned char flg; /* 0 = ワイルドなし, 0xFF = ファイル指定なし */
    unsigned char drive; /* ドライブ番号 (A = 0, B = 1, ...) */
    unsigned char path [65]; /* パス名 (ルート = '¥', サブ = '¥test¥') */
    unsigned char name1 [8]; /* ファイル名 (先頭 8 文字) */
    unsigned char ext [3]; /* 拡張子 */
    unsigned char name2 [10]; /* ファイル名 (9 文字目以降) */
};
```

**戻り値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) でエラー。

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

main()
{
    struct NAMESTBUF info;
    NAMEST("A:\WINDOWS\SYSTEM32\cmd.exe", &info);
    printf("drive = %c\n", info.drive);
    printf("path = %s\n", info.path);
    printf("name = %s\n", info.name);
    printf("ext = %s\n", info.ext);
}
```

## プログラム例

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

#define NAMELEN 8

main()
{
    struct NAMESTBUF inf; /* 解析情報 */
    char name[NAMELEN + 1];
    char ext[EXTLEN + 1];

    if (NAMESTS("A:¥¥CONFIG.SYS", &inf) == 0) {
        printf("flg : %d¥n", inf.flg);
        printf("drive: %d¥n", inf.drive);
        printf("path : %s¥n", inf.path);

        strncpy(name, inf.name1, NAMELEN);
        name[NAMELEN] = '¥0';
        strncpy(ext, inf.ext, EXTLEN);
        ext[EXTLEN] = '¥0';

        printf("name1: %s¥n", name);
        printf("ext : %s¥n", ext);
        printf("name2: %s¥n", inf.name2);
    }
}
```

# NEWFILE

レベル 1

**書 式** # include <doslib.h>

```
int NEWFILE (file, atr) ;  
unsigned char *file; /* ファイル名格納領域へのポインタ */  
int atr; /* ファイル属性 */
```

**機 能** ファイルを新規に作成します。

CREATE 関数と違うのは、指定したファイルがすでに存在していた場合にエラーを返すことです。

atr には、作成するファイルの属性を指定します。

ファイル属性の指定方法は CREATE 関数と同じなので、そちらを参照してください。

**戻 り 値** 作成したファイルのファイルハンドルを返します。

負の数ならエラーを表します。

特に、指定したファイルがすでに存在していた場合には、エラー (-80) を返します。

**参 照 関 数** CREATE

# NFILES

## レベル 1

**書式** #include <doslib.h>

```
int NFILES (buffer) ;
struct FILBUF *buffer; /* ファイル情報格納領域へのポインタ */
struct FILBUF {
    unsigned char OS[21]; /* Human68k 内部で使います */
    unsigned char atr; /* ファイル属性 */
    unsigned short time; /* ファイルの時刻 */
    unsigned short date; /* ファイルの日付 */
    unsigned int filelen; /* ファイルの長さ */
    unsigned char name[23]; /* ファイル名 */
};
```

**機能** FILES で得た buffer を指定して、次のファイルを検索します。

**戻り値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) でエラーを表します。

**参照関数** FILES

**プログラム例**

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

main()
{
    struct FILBUF inf; /* ファイル情報 */
    int atr; /* ファイル属性 */

    atr = 0x20; /* 通常ファイル */

    if (FILES(&inf, " *.*", atr) >= 0) { /* 1つめがあれば */
        printf("%s\n", inf.name); /* それを表示 */

        while (NFILES(&inf) >= 0) /* 2つめ以降の間 */
            printf("%s\n", inf.name); /* 表示する */
    }
}
```

# ONTIME

レベル 0

**書式** #include <iocslib.h>

int ONTIME ( );

**機能** 電源投入、またはリセットしてから現在までの時間を返します。

**戻り値** 0秒~23時59分59秒99までの値 (0~8639999) を返します。  
この値は1日を超えるとリセットされます。

## プログラム例

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

main()
{
    int bin; /* 時間(2進数) */
    int h, m, s, ss;

    bin = ONTIME(); /* BIOS動作時間の通知 */

    h = bin / 3600 / 100;
    m = (bin - (h * 3600 * 100)) / 6000;
    s = (bin - (h * 3600 * 100) - (m * 6000)) / 100;
    ss = bin % 100;

    printf("ontime = %d:%d:%d.%d\n", h, m, s, ss);
}
```

# OPEN

## レベル 1

**書 式** #include <doslib.h>

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

### 機 能

fileで指定したファイルをオープンします。  
modeには、次のアクセスモードを指定します。

- 0x000 : 読み込み
- 0x001 : 書き込み
- 0x002 : 読み込み/ 書き込み
- 0x100 : 読み込み  
(辞書用の特殊ハンドルを返す。ユーザーは使用禁止)
- 0x101 : 書き込み  
(辞書用の特殊ハンドルを返す。ユーザーは使用禁止)
- 0x102 : 読み込み/ 書き込み  
(辞書用の特殊ハンドルを返す。ユーザーは使用禁止)

### 戻 り 値

ファイルハンドルを返します。  
負の数ならエラーを表します。

# OPEN\_PR

レベル 1

## 書式

```
# include <doslib.h>

int OPEN_PR (name, counter, usp, ssp, sr, pc, buff, sleep_time) ;
unsigned char *name; /* スレッドの名前へのポインタ */
int counter; /* タスクの実行間隔の制御パラメータ */
int usp; /* タスク起動時の usp の初期値 */
int ssp; /* タスク起動時の ssp の初期値 */
int sr; /* タスク起動時の sr の初期値 */
int pc; /* タスク起動時の実行開始アドレス */
struct PRCCTRL { /* タスク間通信のための領域 */
    long length; /* データバッファのバイト数 */
    unsigned char *buf_ptr; /* データバッファへのポインタ */
    unsigned short command; /* コマンドバッファ */
    unsigned short your_id; /* 相手の ID のバッファ
        (-1 で通信許可) */
} *buff;
long sleep_time; /* 待ち時間 */
```

## 機能

バックグラウンドタスクを登録します。

登録したスレッドは SLEEP 状態になります。

name には、スレッドの名前を表す 15 文字以内の文字列へのポインタを指定します。

もし、すでに同じ名前スレッドが起動していたらエラーとなります。

counter には、タスクを 1 回実行するために、タスク切り替えのためのタイマー割り込みの発生を何回カウントするか、その回数を指定します。

counter の値は 2~255 です。

0 または 1 が指定された場合は 2 として扱います。

usp, ssp, sr, pc は、それぞれタスクを実行するときのレジスタの初期値です。システム用のスタックは 6 キロバイト必要です。

また、sr は 0 または 0x2000 を指定し、ユーザーモードかスーパーバイザーモードかどちらかを選択します。

その他のレジスタの初期値は全て 0 です。

buff には、タスク間通信用のバッファへのポインタを指定します。

sleep\_time には、起動するまでの待ち時間をミリ秒単位で指定します。

0 を指定すると、永久に SLEEP します。

0112

OPEN\_PR 関数により、バックグラウンドタスクを登録した場合には、KEEP\_PR 関数で常駐終了します。

スレッドをメモリ上から消去したい場合は KILL\_PR 関数を使用します。

**戻り値**

スレッドが登録できたならば、スレッドの ID を返します。

負の数ならエラーを表します。

同じ名前のスレッドがすでに起動していた場合は -27 を、これ以上スレッドを起動できない場合は -29 を返します。

**参照関数**

KILL\_PR, GET\_PR, SUSPEND\_PR, SEND\_PR, TIME\_PR, CHANGE\_PR

# OPMINTST

レベル 0

**書 式**

#include &lt;iocslib. h&gt;

```
int OPMINTST (ADDRESS) ;  
unsigned char *ADDRESS ; /* 割り込み処理アドレス */
```

**機 能**

FM 音源 IC による割り込みを設定します (MFP のみ制御します)。

ADDRESS には割り込み処理アドレスを指定します。

0 を指定すると割り込み禁止にします。

なお、FM 音源デバイスドライバを組み込んでいる場合は、割り込みを変更してはいけません。

**戻 り 値**

割り込みが設定されたときは 0、すでに使用中の場合には 0 以外の値を返します。

# OPMSET

レベル 0

**書 式** #include <iocslib.h>

```
void OPMSET (ADDRESS, DATA) ;  
int ADDRESS; /* 出力アドレス */  
int DATA; /* 書き込むデータ */
```

**機 能** FM 音源 (YM2151) にデータを書き込みます。  
書き込みは FM 音源内部のビジーフラグの状態に従って行われます。  
なお、FM 音源デバイスドライバが曲を演奏している時は、データを書き込んではいけません。

**戻り値** 戻り値はありません。

# OPMSNS

レベル 0

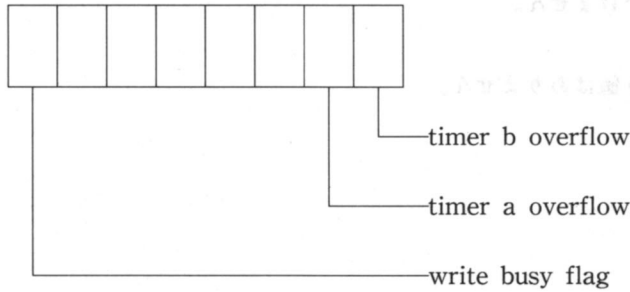
**書 式** #include <iocslib.h>

int OPMSNS ( );

**機 能** FM 音源 (YM2151) のステータスを読み込みます。

**戻 り 値** FM 音源 (YM2151) のステータスを返します。

7 6 5 4 3 2 1 0



0 : 書き込み可

1 : 動作中

# OS\_CUROF

レベル 0

**書 式** #include <iocslib. h>

```
void OS_CUROF ( );
```

**機 能** カーソルを消します (esc[>5h を出力したのと同じ)。

**戻 り 値** 戻り値はありません。

# OS\_CURON

レベル 0

**書 式** #include <iocslib. h>

void OS\_CURON ( ) ;

**機 能** カーソルを表示します (esc[>5l を出力したのと同じ)。

**戻り値** 戻り値はありません。

# OSNS232C

レベル 0

**書 式** #include <iocslib.h>

```
int OSNS232C ( );
```

**機 能** RS-232C 送信が可能か否かチェックします。

**戻 り 値** 戻り値が 0 のときは送信できません。

戻り値が 0 以外のときは送信できる (バッファは空で、XON でもない) ことを示します。

## プログラム例

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

main()
{
    void sub(char *);

    if (OSNS232C() == 0) {
        printf("RS-232C 送信ができません\n");
        exit(1);
    }
    sub("hello\n");
}

static void sub(char *str)
{
    while (*str) {
        while (OSNS232C() == 0)
            ;
        OUT232C(*str++);
    }
}
```

# OUT232C

レベル 0

**書 式** #include <iocslib.h>

```
void OUT232C (DATA) ;  
int DATA; /* 送信データ */
```

**機 能** RS-232C への送信を行います。

**戻 り 値** 戻り値はありません。

## プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
#include <iocslib.h>  
  
main()  
{  
    void sub(char *);  
  
    if (OSNS232C() == 0) {  
        printf("RS-232C送信ができません\n");  
        exit(1);  
    }  
    sub("hello\n");  
}  
  
static void sub(char *str)  
{  
    while (*str) {  
        while (OSNS232C() == 0)  
            ;  
        OUT232C(*str++);  
    }  
}
```

# OUTLPT

## レベル 0

**書式** #include <iocslib.h>

```
void OUTLPT (DATA);
int DATA; /* 出力データ */
```

**機能** プリンタへの出力を行います (漢字処理等をしないで直接出力する)。

**戻り値** 戻り値はありません。

### プログラム例

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

#define MAXLINE 66 /* 印字可能行数 */
#define MAXWIDTH 132 /* 印字可能文字数 */

main()
{
    void sub(char *);

    if (INIT_PRN(MAXLINE - 1, MAXWIDTH - 1) == 0) {
        printf("プリンタへの出力はできません\n");
        exit(1);
    }
    sub("hello#r#n");
}

static void sub(char *str)
{
    while (*str) {
        while (SNSPRN() == 0)
            ;
        OUTLPT(*str++);
    }
}
```

# OUTPRN

レベル 0

**書 式** #include <iocslib.h>

```
void OUTPRN (DATA) ;  
int DATA; /* 出力データ */
```

**機 能** プリンタへの出力を行います (シフト JIS で漢字処理あり)。

**戻 り 値** 戻り値はありません。

## プログラム例

```
#include <stdio.h>  
#include <stdlib.h>  
#include <iocslib.h>  
  
#define MAXLINE 66 /* 印字可能行数 */  
#define MAXWIDTH 132 /* 印字可能文字数 */  
  
main()  
{  
    void sub(char *);  
  
    if (INIT_PRN(MAXLINE - 1, MAXWIDTH - 1) == 0) {  
        printf("プリンタへの出力はできません\n");  
        exit(1);  
    }  
    sub("こんにちは\n");  
}  
  
static void sub(char *str)  
{  
    while (*str) {  
        while (SNSPRN() == 0) ;  
        OUTPRN(*str++);  
    }  
}
```

# PAINT

レベル 0

**書式** #include <ioclib. h>

```
int PAINT (paintptr) ;
struct PAINTPTR *paintptr ;

struct PAINTPTR {
    short x ; /* X座標 */
    short y ; /* Y座標 */
    unsigned short color ; /* パレットコード */
    unsigned char *buf_start ; /* 作業領域先頭アドレス */
    unsigned char *buf_end ; /* 作業領域終了アドレス */
} ;
```

**機能** グラフィック画面のペイントを実行します。

**戻り値** 終了コードを返します。

0 : 正常終了

-1 : グラフィックは使用不可

# PATHCHK

レベル 1

**書 式** #include <doslib. h>

```
int PATHCHK (file, comline, envptr) ;
unsigned char *file; /* ファイル名格納領域へのポインタ */
struct COMLINE *comline; /* コマンドライン名格納領域へのポインタ */
unsined char *envptr; /* 環境文字列へのポインタ */

struct COMLINE {
    unsigned char len; /* コマンドラインの長さ */
    unsigned char buffer [255]; /* コマンドライン文字列 */
};
```

**機 能**

envptr の環境から path をサーチし、file のコマンド行をファイル名（ドライブ名、パス名つき）とコマンドラインに分けて、それぞれ file と comline が示す領域に格納します。

file は 90 バイト以上、comline は 256 バイト以上必要です。envptr に 0 を指定すると、呼び出し側プログラムの環境をサーチします。

このファンクションコールを実行後、comline と file を指定して、LOADEXEC、または LOAD を実行した場合、すでに通ったパスであれば、どのパスにおいても簡単に実行することができます。

envptr はグローバル変数 environ が指す C 独自の環境とはちがいます。OS の環境の構造をしたポインタでなければなりません。0 以外を指定する場合は注意してください。

**戻 り 値**

負の数ならエラーを表します。

**参 照 関 数**

LOADEXEC, LOAD, LOADONLY, EXECONLY

## プログラム例

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

main()
{
    struct COMLINE comline;      /* コマンドライン */
    char *file[90];             /* ファイル名 */
    char *envptr = NULL;        /* 環境文字列 */

    file = "A:¥¥BIN¥¥ED.X A:¥¥CONFIG.SYS";

    if (PATHCHK(file, &comline, envptr) < 0)
        printf("コマンドが見つかりません。¥n")

    printf("file    = %s¥n", file);
    printf("len     = %d¥n", comline.len);
    printf("comline = %s¥n", comline.buffer);
}
```

# POINT

レベル 0

**書 式** #include <iocslib. h>

```
int POINT (pointptr) ;  
struct POINTPTR *pointptr ;
```

```
struct POINTPTR {  
    short x; /* X座標 */  
    short y; /* Y座標 */  
    unsigned short color; /* パレットコード */  
};
```

**機 能** グラフィック画面の指定座標のドットのパレットコードを調べます。

**戻り値** 終了コードを返します。

- 0 : 正常終了
- 1 : グラフィックは使用不可

正常終了した場合には、パレットコードが color に格納されます。

# PRINT

## レベル 1

**書 式** #include <doslib. h>

```
void PRINT (msgptr) ;  
unsigned char *msgptr; /* 表示する文字列へのポインタ */
```

**機 能** ヌル文字で終了する文字列を表示します。  
このときブレークチェックも行います (^C, ^S, ^P, ^N)。

**戻り値** 戻り値はありません。

**参照関数** GETS

# PRNINTST

レベル 0

**書 式** #include <iocslib. h>

```
int PRNINTST (ADDRESS) ;  
/* unsigned char *ADDRESS; /* 割り込み処理アドレス */
```

**機 能** プリンタの割り込みを設定します。

ADDRESS には、割り込み処理アドレスを指定します。  
0 を指定すると割り込み禁止にします。

**戻 り 値** 割り込みが設定されたときは 0、すでに使用中の場合には 0 以外の値を返します。

# PRNOUT

レベル 1

**書 式** #include <doslib. h>

```
void PRNOUT (code) ;  
int code; /* 出力する文字 */
```

**機 能** プリンタへ1文字出力します。

また、このときブレークチェックも行います(Ⓒ)。

**戻 り 値** 戻り値はありません。

**参 照 関 数** PRNSNS, fputc

# PRNSNS

レベル 1

**書 式** #include <doslib. h>

```
int PRNSNS ( ) ;
```

**機 能** プリンタへの出力の可・不可を調べます。

**戻 り 値** 出力不可のときには0、出力可能ならば0以外を返します。

## PSET

## レベル 0

**書 式** #include <iocslib. h>

```
int PSET (psetptr) ;
```

```
* * * * * struct PSETPTR *psetptr ;
```

```
struct PSETPTR {
    short x ; /* X座標 */
    short y ; /* Y座標 */
    unsigned short color ; /* パレットコード */
};
```

**機 能** グラフィック画面の指定座標に、指定したパレットコードで点を描きます。

**戻 り 値** 終了コードを返します。

0 : 正常終了

-1 : グラフィックは使用不可

# PSPSET

レベル 1

**書 式** #include <doslib. h>

```
void PSPSET (pspadr) ;  
struct PDBADR * pspadr ; /* プロセス管理情報格納領域へのポインタ */
```

**機 能**

pspadr で示すアドレスに、プロセス管理情報を書き込みます。  
プロセス管理情報の内容は次の通りです。

```
struct PDBADR {  
    unsigned int env ; /* 環境アドレス */  
    unsigned int exit ; /* 終了アドレス */  
    unsigned int ctrlc ; /* CTRL-C アドレス */  
    unsigned int errexit ; /* エラー中止アドレス */  
    unsigned int comline ; /* コマンドラインアドレス */  
    unsigned char handle [12] ; /* ファイルハンドラ使用状況 */  
    unsigned int bss ; /* bss 先頭アドレス */  
    unsigned int heap ; /* heap 先頭アドレス (= bss) */  
    unsigned int stack ; /* 初期 stack アドレス (= heap END) */  
    unsigned int usp ; /* 親プロセスの USP レジスタ値 */  
    unsigned int ssp ; /* 親プロセスの SSP レジスタ値 */  
    unsigned short sr ; /* 親プロセスの SR レジスタ値 */  
    unsigned short abort_sr ; /* アボート時の SR レジスタ値 */  
    unsigned int abort_ssp ; /* アボート時の SSP レジスタ値 */  
    unsigned int trap10 ; /* POWER OFF or RESET */  
    unsigned int trap11 ; /* STOP キー (HDOFF 処理) 他 */  
    unsigned int trap12 ; /* COPY キー (HCOPY 処理) 他 */  
    unsigned int trap13 ; /* CTRL+C キーでのブレークフラグセット */  
    unsigned int trap14 ; /* エラー表示・処理選択(中止・再実行・無視) */  
    unsigned int osflg ; /* 起動プロセスフラグ (0 = 子プロセスから、  
        1 = OS から) */  
    unsigned char reserve [28] ; /* システムリザーブ */  
    unsigned char exe_path [68] ; /* ドライブ名およびパス名 */  
    unsigned char exe_name [24] ; /* コマンドファイル名 */  
};
```

pspadr はメモリ管理関数 MALLOC からの戻り値で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 <stdlib.h>
#include <doslib.h>

main()
{
    struct PDBADR *psp; /* プロセス管理情報 */
    psp = (struct PDBADR *)MALLOC(sizeof(struct PDBADR));
    PSPSET(psp);
}
```

# PUTCHAR

## レベル 1

**書 式** #include <doslib. h>

```
void PUTCHAR (code) ;  
int code; /* 表示する文字 */
```

**機 能** 指定された文字コードを表示します。  
また、このときブレークチェックも行います (^C, ^S, ^P, ^N)。

**戻 り 値** 戻り値はありません。

**参 照 関 数** GETCHAR, INPOUT, PUTC

# PUTGRM

レベル 0

**書 式** #include <ioclib.h>

```
int PUTGRM (putptr) ;  
struct PUTPTR * putptr ;  
  
struct PUTPTR {  
    short x1; /* 始点の X 座標 */  
    short y1; /* 始点の Y 座標 */  
    short x2; /* 終点の X 座標 */  
    short y2; /* 終点の Y 座標 */  
    unsigned char *BUF_START; /* バッファ先頭アドレス */  
    unsigned char *BUF_END; /* バッファ終了アドレス */  
};
```

**機 能**

グラフィック画面の書き出しを行います。  
バッファには、データがパックされた型で格納されています。

**戻 り 値**

終了コードを返します。

- 0 : 正常終了
- 1 : グラフィックは使用不可

# RASSIGN

レベル 1

**書式** # include <doslib. h>

```
int RASSIGN (buff1) ;
unsigned char *buff1; /* 仮想ドライブ名、仮想ディレクトリ名へのポインタ */
```

**機能** buff1 で指定された仮想ドライブ名、仮想ディレクトリ名をキャンセルします。buff1 は、MAKEASSIGN 関数での 1 番目のパラメータと同じ内容のパラメータを指定してください。

**戻り値** 負の数ならエラーを表します。

**参照関数** GETASSIGN, MAKEASSIGN

# READ

## レベル 1

**書式** #include <doslib.h>

```
int READ (fileno, buffer, size) ;
int fileno ; /* 該当ファイルのファイルハンドル */
unsigned char *buffer ; /* 読み込みデータ格納領域へのポインタ */
int size ; /* 読み込みバイト数 */
```

**機能** fileno で指定するファイルハンドルから、size で指定するバイト数を、buffer が指す領域へ読み込みます。

**戻り値** 読み込んだバイト数を返します。  
これが負の数ならばエラーコードを示します。

**参照関数** WRITE

```

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

main()
{
    int fd;
    char *old = "A:YCONEID.BAT";
    char *new = "A:YCONEID.BAT";

    if (fd = open(new, O_RDONLY) < 0)
        return (0);
    else
        close(fd);
    return (1);
}

static int
exec(char *file)
{
    if (exec(file) < 0)
        return (1);
    else
        return (0);
}

```

# RENAME

レベル 1

**書 式** #include <doslib.h>

```
int RENAME (old, new) ;
unsigned char *old; /* 旧ファイル名へのポインタ */
unsigned char *new; /* 新ファイル名へのポインタ */
```

**機 能** oldで指定したファイルの名前を、newで示す名前に変更します。  
オープンされているファイルに対して実行した場合はエラーになります。

**戻り値** 正の数（0も）で正常終了、負の数（-1以外も）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は、0以外の正の数の場合があります。

**参 照 関 数** MOVE

## プログラム例

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

main()
{
    int exist(char *);
    char *old = "A:¥¥CONFIG.SYS";
    char *new = "A:¥¥CONFIG.BAK";

    if (!exist(old))
        RENAME(new, old); /* 変更(2回目) */
    else
        RENAME(old, new); /* 変更(1回目) */
}

static int exist(char *file)
{
    int fd;

    if ((fd = open(file, O_RDONLY)) < 0)
        return (0);
    else {
        close(fd);
        return (1);
    }
}
```

# RMACNV

レベル 0

**書 式** #include <iocslib.h>

```
int RMACNV (CODE, W_POINTER, A_POINTER) ;
int CODE; /* 文字コード */
unsigned char *W_POINTER; /*ワークポインタ */
unsigned char *A_POINTER; /*答えのバッファへのポインタ */
```

**機 能** ローマ字変換を行います。

CODE には文字コードを指定します。

W\_POINTER にはワークポインタ、A\_POINTER には答えのバッファを指すポインタを指定します。

**戻 り 値** 変換ステータスを返します。

0 : 変換の途中

-1 : 変換不可能

上記以外 : 変換文字数を示します。(W\_POINTER に変換途中の文字が残っている可能性もあります)。

## プログラム例

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

main()
{
    char work[10];
    char ansr[10];
    int code; /* 文字コード */

    work[0] = ansr[0] = '\0';
    code = 'A';

    RMACNV(code, work, ansr);

    printf("%c → %c\n", code, ansr[0]);
}
```

# RMDIR

## レベル 1

**書式** #include <doslib.h>

```
int RMDIR (file);  
unsigned char *file; /* ディレクトリ名格納領域へのポインタ */
```

**機能** ディレクトリを削除します。  
ただし、カレントディレクトリや、空でないディレクトリは削除できません。

**戻り値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) 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 以外の正の数の場合があります。

**参照関数** MKDIR, CHDIR

```
#include <stdio.h>  
#include <doslib.h>  
  
int  
main()  
{  
    char work[10];  
    char ansr[10];  
    int code;  
  
    work[0] = ansr[0] = '\0';  
    code = 'A';  
  
    RMDIR(code, work, ansr);  
  
    printf("Xc - %c\n", code, ansr[0]);  
}
```

# ROMVER

レベル 0

**書式** #include <iocslib.h>

int ROMVER ( ) ;

**機能** ROM のバージョンと作成年月日を調べます。**戻り値** ROM のバージョンと作成年月日を、下に示すフォーマットで返します。

VVVVVVVV YYYYYYYY MMMMMMMM DDDDDDDD

VVVVVVVV : バージョン (0x10)

YYYYYYYY : 年 (0x87)

MMMMMMMM : 月 (0x05)

DDDDDDDD : 日 (0x07)

**プログラム例**

```

#include      <stdio.h>
#include      <iocslib.h>

main()
{
    int      verymd;          /* バージョン・年月日 */

    verymd = ROMVER();

    printf("ver.   = %x\n", (verymd >> 24) & 0xff);
    printf("year  = %x\n", (verymd >> 16) & 0xff);
    printf("month = %x\n", (verymd >> 8) & 0xff);
    printf("day   = %x\n", (verymd >> 0) & 0xff);
}

```

# SCROLL

レベル 0

**書 式** #include <iocslib. h>

```
void SCROLL (MODE, XDOT, YDOT) ;  
int MODE; /* モード */  
int XDOT; /* Xドット座標 */  
int YDOT; /* Yドット座標 */
```

**機 能** テキスト/グラフィックの表示座標の設定を行います。  
MODEには次の値を設定します。

- 0 : グラフィックスクリーン0 設定
- 1 : グラフィックスクリーン1 設定
- 2 : グラフィックスクリーン2 設定
- 3 : グラフィックスクリーン3 設定
- 8 : テキスト設定

**戻り値** 戻り値はありません。

# SEEK

## レベル 1

**書 式** #include <doslib.h>

```
int SEEK (fileno, offset, mode) ;
int fileno; /* 該当ファイルのファイルハンドル */
int offset; /* ファイルポインタの移動バイト数 */
int mode; /* モード */
```

**機 能** fileno のファイルハンドルで指定されるランダムファイルを、各 mode に従い、offset 分だけファイルポインタを移動します。

モードには次の内容を指定します。

0 : ファイルの先頭

1 : ファイルの現在位置

2 : ファイルの終り

**戻 り 値** 現在のファイルポインタの先頭からのオフセットを返します。  
負の数ならエラーを表します。

# SEND\_PR

レベル 1

**書 式** # include <doslib.h>

```
int SEND_PR (my_id, your_id, command, buff, length) ;  
int my_id; /* 自分のスレッドの ID */  
int your_id; /* 通信したい相手スレッドの ID */  
int command; /* 通信コマンド */  
unsigned char *buff; /* データバッファへのポインタ */  
long length; /* データバッファのバイト数 */
```

**機 能**

指定した ID のスレッドにコマンドやデータを送り、SLEEP していたら起こします。

my\_id は自分のスレッドの ID です。

your\_id は通信したい相手のスレッドの ID です。

command は通信の内容を表すワードの値で、システムで定められているコマンド以外はそれぞれのスレッド間で定義します。

システムで予約してあるコマンドは 0xFF?? であり、以下に示す値が定義されています。

0xFFFF9 : スレッドを消去するよう要求する

0xFFFFB : 強制 SLEEP 状態から起こすのみ (タスク間通信バッファは変化しない)

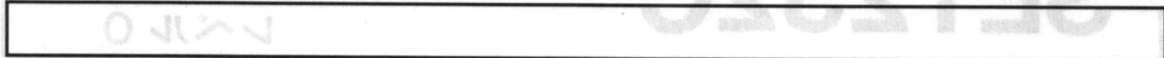
0xFFFFC : SLEEP するよう要求する

起こされたときこのコマンドが送られていたならば、すぐにタスク間通信バッファの your\_id を -1 にして SLEEP すべきである通常、SLEEP しないでタスク間通信バッファを監視している場合に有効

0xFFFFF : 処理が終わったかどうか調べるためのコマンド  
-28 が返れば、まだ処理中であることを示す

タスク間通信バッファの構造体を以下に示します。

```
struct PRCCTRL {  
    long length;  
    unsigned char *buf_ptr;  
    unsigned short command;  
    unsigned short your_id;  
};
```



引数の `your_id` で指定したスレッドの、タスク間通信バッファの `your_id` が `-1` に設定されていれば、書き込み可能です。

`my_id` とコマンドをそれぞれ通信バッファの `your_id`, `command` に設定し、`buff` からのデータを `length` だけ、通信バッファの `buf_ptr` で示されるデータバッファに書き込みます。

そして `length` を通信バッファの `length` に設定します。指定したスレッドが `SLEEP` していたら起こします。

特に、`0xFFFFB` のコマンドは特殊処理され、指定したスレッドを起こすだけで、タスク間通信バッファの `your_id` が `-1` でなくてもよく、その他のバッファも変化しません。

引数の `length` が、タスク間通信バッファの `length` より大きいときはエラーとなります。

また、書き込み不可の場合もエラーとなります。

**戻り値**

負の数ならエラーを表します。

正常終了の場合は `0` を返します。

エラーコードが `0xFFFF00??` の場合は、指定した `id` が不正であり、`??` は `id` の最大値を示します。

エラーコードが `0x80??????` の場合は、指定した `length` が不正であり、`??????` は `length` の最大値を示します。

書き込みエラーの場合は、`-28` を返します。

**参照関数**

`OPEN_PR`, `KILL_PR`, `GET_PR`, `SUSPEND_PR`, `SLEEP_PR`, `TIME_PR`, `CHANGE_PR`

# SET232C

レベル 0

## 書式

```
#include <ioclib.h>
```

```
int SET232C (MODE) ;
```

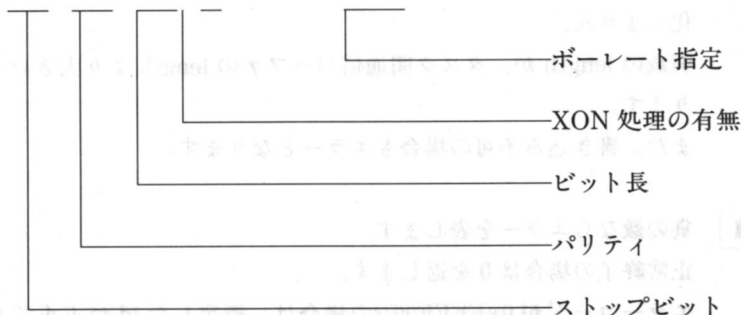
```
int MODE; /* モード */
```

## 機能

RS-232C のモードを設定します。

MODE の内容と、各項目に設定できる値を次に示します。

XX XX XX X 0 00000 XXX



### ● ストップビット

- 01 : 1 ビット
- 10 : 1.5 ビット
- 11 : 2 ビット
- 00 : 2 ビット

### ● パリティ

- 01 : 奇数
- 11 : 偶数
- 00 : パリティなし
- 10 : パリティなし

### ● ビット長

- 00 : 5 ビット以下
- 01 : 6 ビット
- 10 : 7 ビット
- 11 : 8 ビット

SETBLOCK

## ● XON 処理の有無

0 : 処理しない

1 : 処理する

## ● ボーレート指定

000 : 75 ボー

001 : 150 ボー

010 : 300 ボー

011 : 600 ボー

100 : 1200 ボー

101 : 2400 ボー

110 : 4800 ボー

111 : 9600 ボー

## 戻り値

MODE に -1 を指定したときのみ、前のモードを返します。

## プログラム例

```

#include      <iocslib.h>

#define STOP1    0x4000 /* ストップビット = 1bit */
#define PEVEN    0x3000 /* パリティ = 偶数 */
#define BITS8    0x0c00 /* ビット長 = 8ビット */
#define XON_E    0x0200 /* xon/off処理 = あり */
#define B9600    0x0007 /* bps = 9600bps */

main()
{
    int      mode; /* 設定モード */

    mode = STOP1 | PEVEN | BITS8 | XON_E | B9600;
    SET232C(mode);
}

```

# SETBLOCK

レベル 1

**書 式** #include <doslib. h>

```
int SETBLOCK (memptr, newlen) ;
int memptr; /* メモリブロックを指すポインタ */
int newlen; /* 変更後のメモリサイズ */
```

**機 能**

memptr で指定されたメモリブロックのサイズを、newlen に変更します (サイズは、大きくすることも小さくすることもできます)。

memptr は MALLOC が返した値です。

したがって、誤った memptr を指定するとエラーとなります。

newlen に 0x1000000 以上の値を指定したときは、0xFFFFF として処理します。

**戻 り 値**

終了コードを返します。

変更できなかったときには 0x81000000 + 最大バイト数を返します。

0x82000000 は、まったく変更できないことを示します。

```

#include <doslib.h>
#define STOP 0x4000
#define RWEN 0x3000
#define BIT8 0x2000
#define X08 0x1000
#define B0800 0x0000
/* ... */
main()
{
    int mode;
    mode = STOP | RWEN | BIT8 | X08 | B0800;
    SET32G(mode);
}

```

# SETDATE

レベル 1

**書式** #include <doslib.h>

```
int SETDATE (date) ;
int date; /* 設定する日付 */
```

**機能**

日付を設定します。  
date の形式は次の通りです。

```
yyyyyyym mmmddddd
```

ビット 0~4 (ddddd) : 日 (1~31)

ビット 5~8 (mmmm) : 月 (1~12)

ビット 9~15 (yyyyyy) : 年 (0~99、1980 年からの相対年数)

**戻り値**

正の数 (0 も) で正常終了、負の数 (-1 以外も) でエラー。

**参照関数**

GETTIM2, SETTIM2, GETDATE, GETTIME, SETTIME

**プログラム例**

```
#include <doslib.h>

main()
{
    int date; /* 日付 */
    int yy = 7; /* 年 */
    int mm = 10; /* 月 */
    int dd = 5; /* 日 */

    date = (yy << 9) | (mm << 5) | (dd << 0);

    SETDATE(date);
}
```

# SETENV

レベル 1

**書式** #include <doslib. h>

```
int SETENV (name, env, buffer) ;
unsigned char *name; /* 環境変数へのポインタ */
unsigned char *env; /* 環境文字列へのポインタ */
unsigned char *buffer; /* 環境変数格納領域へのポインタ */
```

**機能**

env で指定された環境に変数を設定します。  
 env に 0 を指定すると、親プロセスの環境に変数を設定します。  
 buffer に 0 を指定すると、name の環境変数を消去します。  
 なお、環境変数には 255 バイトまでしか設定できません。

**戻り値**

負の数ならエラーを表します。

**参照関数**

GETENV

```
#include <doslib.h>
int main()
{
    int date;
    int yy = 7;
    int mm = 10;
    int dd = 0;

    date = (yy << 9) | (mm << 5) | dd << 0;

    SETENV(date);
}
```

# SETPDB

レベル 1

**書式** #include <doslib.h>

```
int SETPDB (pdbadr) ;
struct PDBADR *pdbadr; /* プロセスのアドレス */
```

**機能** pdbadr で示すプロセスに制御をわたします。  
 pdbadr は、プログラムの先頭アドレス-0xF0 です。  
 なお、pdbadr は GETPDB が返したポインタでなければなりません。

**戻り値** 前の pdbadr の値を返します。

**参照関数** GETPDB

```
#include <doslib.h>
main()
{
  int time;
  int bh = 15;
  int cm = 20;
  int ee = 30;

  time = bh << 16 | cm << 8 | ee << 0;
  SETPDB(time);
}
```

# SETTIM2

レベル 1

**書 式** #include <doslib.h>

```
int SETTIM2 (time) ;  
int time; /* 設定する時刻 */
```

**機 能** 時刻を time の値に設定します。  
time の形式は次の通りです。

```
00000000 000hhhhh 00mmmmmm :00sssss  
ビット 0~5      (sssss) : 秒 (0~59)  
ビット 8~13    (mmmmmm) : 分 (0~59)  
ビット 16~20   (hhhhh)  : 時 (0~23)
```

**戻 り 値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) でエラー。

**参 照 関 数** GETTIM2, GETDATE, SETDATE, GETTIME, SETTIME

## プログラム例

```
#include      <doslib.h>  
  
main()  
{  
    int    time;          /* 設定する時刻 */  
    int    hh = 12;      /* 時 */  
    int    mm = 50;      /* 分 */  
    int    ss = 30;      /* 秒 */  
  
    time = (hh << 16) | (mm << 8) | (ss << 0);  
    SETTIM2(time);  
}
```

# SETTIME

レベル 1

**書 式** #include <doslib.h>

```
int SETTIME (time) ;
int time; /* 設定する時刻 */
```

**機 能** 時刻を設定します。  
time の形式は次の通りです。

hhhhmmmm mmmsssss

ビット 0~4 (sssss) : 秒 (0~29、実際の値は2を乗ずる)

ビット 5~10 (mmmmmm) : 分 (0~59)

ビット 11~15 (hhhhh) : 時 (0~23)

**戻り値** 正の数 (0 も) で正常終了、負の数 (-1 以外も) でエラー。**参照関数** GETTIM2, SETTIM2, GETDATE, SETDATE, GETTIME**プログラム例**

```
#include <doslib.h>

main()
{
    int time; /* 時刻 */
    int hh = 10; /* 時 */
    int mm = 25; /* 分 */
    int ss = 20 / 2; /* 秒 */

    time = (hh << 11) | (mm << 5) | (ss << 0);
    SETTIME(time);
}
```

# SFTJIS

レベル 0

**書式** #include <iocslib.h>

```
int SFTJIS (CODE) ;
int CODE; /* シフト JIS コード */
```

**機能** シフト JIS コードを JIS コードに変換します。

**戻り値** JIS 漢字コードを返します。  
ただし、上位 16 ビットが 0xFFFF ならエラーが発生したことを示します。

## プログラム例

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

main()
{
    int ms; /* シフト JIS コード */
    int jis; /* JIS 漢字コード */

    ms = 0x9872;

    jis = SFTJIS(ms);

    printf("0x%04x(sft) : 0x%04x(jis)%n", ms, jis);
}
```

# SKEY\_MOD

レベル 0

**書 式** #include <ioclib. h>

```
int SKEY_MOD (MODE, X, Y) ;
int MODE; /* モード */
int X; /* X座標 */
int Y; /* Y座標 */
```

**機 能** ソフトキーボードの制御を行います。  
MODE には次の内容を指定します。

- 0 : ソフトキーボードの消去
- 1 : ソフトキーボード表示 (表示状態では無効)
- 2 : ソフトキーボード表示状態のチェック
- 1 : ソフトキーボード自動制御 (右ボタンで表示/消去)

X には X 座標、Y には Y 座標を指定します。

なお、MODE が 1 以外では、X、Y は無意味となります。

**戻り値** ソフトキーボードの状態を返します。

- 0 : 消去状態
- 1 : 表示状態

# KEY\_MOD

011111

## プログラム例

```

#include      <stdio.h>
#include      <iocslib.h>

#define SKEYOFF 0      /* ソフトキーボード消去 */
#define SKEYON  1      /* ソフトキーボード表示 */
#define SKEYCHK 2      /* ソフトキーボード状態チェック */
#define XPOS    12     /* X座標 */
#define YPOS    10     /* Y座標 */

main(int argc, char *argv[])
{
    void    usage(void);
    int     mode;

    if (argc == 1)
        mode = SKEY_MOD(mode, XPOS, YPOS);
    else if (argc == 2) {
        if (strcmp(argv[1], "on") == 0)
            SKEY_MOD(SKEYON, XPOS, YPOS);
        else if (strcmp(argv[1], "off") == 0)
            SKEY_MOD(SKEYOFF, XPOS, YPOS);
        else
            usage();
    }
    else
        usage();
    exit(0);
}

static void    usage()
{
    fprintf(stderr, "Usage: key_mod [{on|off}]%n");
    exit(1);
}

```

# SKEYSET

## レベル 0

**書 式** #include <iocslib.h>

```
void SKEYSET (KEYCODE) ;
int KEYCODE; /* キーコード */
```

**機 能**

KEYCODE で指定されたキーを押されたことにします。  
KEYCODE には、BITSNS 関数で指定するキーコードグループを8倍したものに、そのグループのキーのビット位置の番号を加えたものを指定します。  
例えば、'A'ならばグループは3、ビット位置は6なので、 $3 \times 8 + 6 = 0x1E$  を指定します。

**戻 り 値**

戻り値はありません。

**プログラム例**

```
#include <iocslib.h>

main()
{
    int    keycode;      /* キーコード */
    keycode = 0x1E;

    SKEYSET(keycode);   /* キーコード入力 */
}
```

# SLEEP\_PR

レベル 1

**書 式** # include <doslib.h>

```
long SLEEP_PR (time) ;  
long time; /* 待ち時間 (単位はミリ秒) */
```

**機 能** SLEEP 状態に入ります。

time には、待ち時間を指定します。

0 を指定した場合は永久に SLEEP します。

SLEEP 状態に入ったスレッドは、SEND\_PR 関数を用いて強制的に起こすことができます。

SEND\_PR 関数のコマンドが 0xFFFB の場合、タスク間通信バッファは変化しません。

それ以外のコマンドの場合はバッファに次のように設定されます。

```
struct PRCCTRL {  
    long length; /* データバッファに書き込まれたバイト数 */  
    unsigned char *buf_ptr; /* データバッファの先頭アドレス (変化しない) */  
    unsigned short command; /* コマンド番号 */  
    unsigned short your_id; /* 起こしてくれたスレッドの ID */  
} *buff;
```

SLEEP する前に、データバッファの内容を処理してからデータバッファのアドレスとバイト数を設定し、buff->your\_id に -1 を設定してください。

そうすることで、他のアドレスからの通信を許可することになります。

SLEEP していない時でも、SEND\_PR 関数によりデータが送られる場合があります。

その場合 SLEEP するとすぐ起こされ、設定した待ち時間を戻り値として返します。

**戻り値** 待ち時間が経過して自分で起きた場合は、-1 を返します。

タスク間通信バッファの内容は変化しません。

待ち時間が経過したが、SUSPEND\_PR 関数で止められていたため SEND\_PR 関数で起こされた場合は、-2 を返します。

タスク間通信バッファは、SEND\_PR 関数により変化します。

戻り値が -1 でも -2 でもない場合、残り時間をミリ秒単位で返します。

バッファの内容は、SEND\_PR 関数により変化します。

**参照関数** OPEN\_PR, KILL\_PR, GET\_PR, SUSPEND\_PR, TIME\_PR, CHANGE\_PR

# S\_MALLOC

レベル 1

**書 式** # include <doslib.h>

```
int S_MALLOC (md, size) ;  
int md; /* メモリ割り当てのモード */  
int size; /* 確保したい領域のサイズ */
```

**機 能**

sizeで指定されたバイト数のメモリ領域を、メインのメモリ管理から確保し、そのポインタを返します。

mdには、以下に示すメモリの割り当て方法を指定します。

0: 下位の方から探し、割り当てる

1: 指定されたサイズを満たすメモリブロックのうち最小のブロックを割り当てる

2: 上位の方から探し、割り当てる

sizeに0x1000000以上の値を指定した時は、0xFFFFFFFFとして処理します。

S\_MALLOCは、通常のアプリケーションプログラムで使用してはいけません。S\_MALLOCを使用するプログラムは、OSから起動されて、終了することができないプログラムでなければなりません。

**戻 り 値**

メモリが確保できた時は、確保できた領域を指すポインタを返します。

確保できなかった時は、0x81000000+最大バイト数を返します。

0x82000000は全く確保できないことを表します。

**参 照 関 数**

MALLOC, MFREE, S\_MFREE, S\_PROCESS

# S\_MFREE

レベル 1

**書式** # include <doslib.h>

```
int S_MFREE (memptr) ;  
int memptr ; /* メモリブロックを指すポインタ */
```

**機能**

メインのメモリ管理のポインタを memptr で指定して、メモリブロックを開放します。

memptr は S\_MALLOC 関数が返した値でなければならないので、誤った memptr を指定するとエラーになります。

memptr が S\_PROCESS 関数で指定しているアドレスで、かつそのスレッド ID がカレント ID ならば、自分自身を削除します (KILL\_PR 関数)。

その時、内部のメモリ管理で常駐終了しているプロセスなどは、メインのメモリ管理に連結されます。

この場合、KILL\_PR と同じ動作になりますので、注意してください。

また、S\_MFREE を行う場合は、指定した memptr のメモリブロック中で動作しているプログラムが存在していない状態で行うようにしてください。

**戻り値**

正の数 (0 も) で正常終了、負の数 (-1 以外も) でエラー。

**参照関数**

MALLOC, MFREE, S\_MALLOC, S\_PROCESS

## SNSPRN

レベル 0

**書式** #include <iocslib.h>

int SNSPRN ( );

**機能** プリンタ出力が可能か否かをチェックします。

**戻り値** 0 以外ならばプリンタへの出力は可能です。

0 のときには出力できません。

## プログラム例

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

#define MAXLINE      66      /* 印字可能行数 */
#define MAXWIDTH    132     /* 印字可能文字数 */

main()
{
    void sub(char *);

    if (INIT_PRN(MAXLINE - 1, MAXWIDTH - 1) == 0) {
        printf("プリンタへの出力はできません\n");
        exit(1);
    }
    sub("こんにちは¥r¥n");
}

static void sub(char *str)
{
    while (*str) {
        while (SNSPRN() == 0)
            ;
        OUTPRN(*str++);
    }
}
```

# SPALET

レベル 0

**書 式** #include <iocslib. h>

```
int SPALET (CODE, BLOCK, COLORCODE);
int CODE; /* 垂直帰線期間との同期 */
int BLOCK; /* パレットブロック */
int COLORCODE; /* カラーコード */
```

**機 能** スプライトパレットの設定と読み出しを行います。

CODE には、垂直帰線期間と同期してパレットを変えるか、同期を取らずに変えるかを指定します。

なお、垂直帰線期間を検出せずにパレットコードを変えると、画面にちらつきが生じます。

0x00~0x0F 垂直帰線期間を検出後に設定

0x80~0x8F 検出なし

BLOCK にはパレットブロックを指定します (1~15)。

COLORCODE にはカラーコードを指定します (0~65535)。

-1 を指定した場合には、カラーコードは変更せずにカラーコードの読み出しのみ行います。

**戻 り 値** 設定前のカラーコード、またはエラーコード (-2 = パラメータエラー、パレットブロック 0 をアクセスした) を返します。

# SP\_CGCLR

レベル 0

**書式** #include <iocslib.h>

```
int SP_CGCLR (CODE) ;
int CODE; /* パターンコード */
```

**機能** PCG をクリアします。  
CODE にはパターンコードを指定します。(0~255、16×16 ドットパターン)。

**戻り値** 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)

# SP\_DEFCG

レベル 0

**書 式** #include <iocslib. h>

```
int SP_DEFCG (CODE, SIZE, ADDRESS) ;  
int CODE; /* パターンコード */  
int SIZE; /* パターンサイズ */  
unsigned char *ADDRESS; /* パターンデータバッファのアドレス */
```

**機 能**

PCG の設定を行います。

CODE にはパターンコードを指定します (0~255)。

SIZE にはパターンサイズを指定します。

- 0 : 8×8 ドットパターン
- 1 : 16×16 ドットパターン

ADDRESS にはパターンデータバッファのアドレスを指定します。

このときパターンデータバッファの SIZE は、は次のようになります。

- 0 : 32 バイト
- 1 : 128 バイト

**戻り値**

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)

# SP\_GTPCG

レベル0

**書 式** #include <iocslib. h>

```
int SP_GTPCG (CODE, SIZE, ADDRESS) ;  
int CODE; /* パターンコード */  
int SIZE; /* パターンサイズ */  
unsigned char *ADDRESS; /* パターンデータバッファのアドレス */
```

**機 能** PCG の読み出しを行います。

CODE にはパターンコードを指定します (0~255)。

SIZE にはパターンサイズを指定します。

- 0 : 8×8 ドットパターン
- 1 : 16×16 ドットパターン

ADDRESS には、パターンデータバッファのアドレスを指定します。  
このときパターンデータバッファの SIZE は、次のようになります。

- 0 : 32 バイト
- 1 : 128 バイト

**戻 り 値** 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)

正常終了の場合、パターンデータバッファにデータが格納されます。

# SP\_INIT

レベル 0

**書式** #include <iocslib. h>

int SP\_INIT ( ) ;

**機能** スプライト面の初期化を行います。

**戻り値** 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)

# SP\_OFF

レベル 0

**書 式** #include <iocslib. h>

void SP\_OFF ( ) ;

**機 能** スプライト面の表示を消します。

**戻 り 値** 戻り値はありません。

# SP\_ON

レベル 0

**書 式** #include <iocslib. h>

int SP\_ON ( ) ;

**機 能** スプライト面の表示を行います。

**戻 り 値** 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)

# SP\_REGGT

レベル 0

**書 式** #include <iocslib. h>

```
int SP_REGGT (SPR_NO, X, Y, CODE, PRW) ;

int SPR_NO; /* スプライトプレーン番号 */
int *X; /* X座標 */
int *Y; /* Y座標 */
int *CODE; /* パターンコード */
int *PRW; /* プライオリティ */
```

**機 能**

スプライトレジスタの読み出しを行います。

SPR\_NO にはスプライトプレーン番号を指定します (0~127)。

X には X 座標、Y には Y 座標を返すポインタを指定します (0~1023)。

PRW にはプライオリティを返すポインタを指定します (0~3)。

CODE には次の内容を返すポインタを指定します。

ビット 16~32 : 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)

**戻 り 値**

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)

正常終了の場合、X 座標、Y 座標、パターンコード、プライオリティがそれぞれに格納されます。

# SP\_REGST

レベル 0

**書 式** #include <iocslib.h>

```
int SP_REGST (SPRITENO, MODE, X, Y, CODE, PRW) ;
int SPRITENO; /* スプライトプレーン番号 */
int MODE; /* 垂直帰線期間の検出モード */
int X; /* X座標 */
int Y; /* Y座標 */
int CODE; /* パターンコード */
int PRW; /* プライオリティ */
```

**機 能**

スプライトレジスタの設定を行います。  
SPRITENO には、スプライトプレーン番号を指定します (0~127)。  
MODE には、垂直帰線期間の検出の有無を指定します (最上位ビットが 0 ならば検出後に設定、1 ならば検出なし)。  
X には X 座標、Y には Y 座標をそれぞれ 0~1023 の範囲内で指定します。  
-1 が指定された場合には、設定を変更せずに現在値を使用します。  
CODE には、パターンコードを指定します。  
-1 が指定された場合には、設定を変更せずに現在値を使用します。  
詳しくは、SP\_REGGT 関数を参照してください。  
PRW には、プライオリティを指定します (0~3)。  
-1 が指定された場合には、設定を変更せずに現在値を使用します。

**戻り値**

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)

# S\_PROCESS

レベル 1

**書式** # include <doslib.h>

```
int S_PROCESS (id, start, size, i_len) ;
int id; /* スレッド ID */
int start; /* サブのメモリ管理の先頭アドレス */
int size; /* サブのメモリ管理のバイト数 */
int i_len; /* 先頭ブロックのバイト数 */
```

**機能**

サブのメモリ管理領域を設定します。

idにはスレッド ID を指定します。

start にはサブのメモリ管理の先頭アドレス、size にはその領域のバイト数を指定します。

指定された ID のスレッドでのメモリ管理は、以後この範囲内となります。

i\_len には、新しいメモリ管理の先頭のメモリブロック (必ず確保される) のバイト数を指定します。

また、start からの最初の 16 バイトは、メモリ管理領域として使用されるので破壊されます。

なお、id = 0 (メインスレッド) は、S\_PROCESS 関数で変更することはできません。

start には、S\_MALLOC で返されたメモリブロックのアドレスを指定します。

S\_PROCESS は、通常のアプリケーションプログラムで使用してはいけません。

S\_PROCESS を使用するプログラムは、OS から起動されて、終了することができないプログラムでなければなりません。

**戻り値**

正常終了なら i\_len で指定したバイト数のメモリブロックの先頭を指すポインタを返します。

戻り値が 0xFFFF00?? なら ID エラーです。

戻り値が -14 なら、size が i\_len + 16 より小さいことを示します。

**参照関数**

MALLOC, MFREE, S\_MALLOC, S\_MFREE

# SUPER

## レベル 1

**書 式** #include <doslib.h>

```
int SUPER (stack) ;  
int stack ; /* 切り替えモード */
```

**機 能**

スーパーバイザモード/ユーザーモードの切り替えを行います。  
stack が 0 のときは、USP の値を SSP にセットして、スーパーバイザモードに切り替えます。  
stack が 0 以外のときは、stack を SSP にセットして、ユーザーモードに切り替えます。

**戻 り 値**

stack が 0 のときは前の SSP の値を返します。  
負の数ならエラーを表します。

**プログラム例**

```
#include <doslib.h>  
  
main()  
{  
    /* ユーザモード → スーパーバイザモード */  
    SUPER(0);  
}
```

## SUPER\_JSR

## レベル 1

**書 式** # include <doslib. h>

```
void SUPER_JSR (func, inregs, outregs) ;
void (*func) ( ) ; /* プログラム先頭アドレス */
struct DREGS {
    int d0 ;
    int d1 ;
    int d2 ;
    int d3 ;
    int d4 ;
    int d5 ;
    int d6 ;
    int d7 ;
    int a0 ;
    int a1 ;
    int a2 ;
    int a3 ;
    int a4 ;
    int a5 ;
    int a6 ;
} *inregs ; /* プログラムに渡すレジスタの値 */
struct DREGS *outregs ; /* プログラムが返すレジスタの値 */
```

**機 能**

スーパーバイザ領域のプログラムをサブルーチンコールします。

func は、呼び出したいプログラムのエントリアドレスです。

inregs には、呼び出しプログラムに渡す各レジスタの初期値の格納領域を指すポインタを指定します。

outregs には、呼び出したプログラムから制御が戻るときのレジスタの状態を格納する、バッファの先頭のポインタを指定します。

なお、sr は変化しません。

usp/ssp はどのような値が渡されるか不定なので、スタックを使って引数を渡すことはできません。

なお、サブルーチンコールの結果、暴走やバスエラーが発生しても処理していません。

**戻 り 値**

戻り値はありません。

# SUSPEND\_PR

レベル 1

**書式** # include <doslib.h>

```
int SUSPEND_PR (id) ;  
int id; /* スレッドの ID */
```

**機能** 指定した ID のスレッドを強制的に SLEEP 状態にします。  
SLEEP 状態になったスレッドは SEND\_PR で起こされるまで SLEEP します。

**戻り値** 正常終了なら 0 を返します。  
負の数ならエラーを表します。  
ID がおかしい場合は 0xFFFF00?? を返します。?? が ID の最大値です。  
自分自身を指定した場合は -1 を返します。

**参照関数** OPEN\_PR, KILL\_PR, GES\_PR, SLEEP\_PR, SEND\_PR, TIME\_PR,  
CHANGE\_PR

## SYMBOL

## レベル 0

**書 式** #include <iocslib. h>

```
int SYMBOL (symbolptr) ;
struct SYMBOLPTR {
    short x1; /* X座標 */
    short y1; /* Y座標 */
    unsigned char *string_address; /* 文字列アドレス */
    unsigned char mag_x; /* 横倍率 */
    unsigned char mag_y; /* 縦倍率 */
    unsigned short color; /* パレットコード */
    unsigned char font_type; /* 文字フォントのタイプ */
    unsigned char angle; /* 角度コード */
} *symbolptr;
```

**機 能**

グラフィック画面のシンボルを実行します。

X1、Y1 には、グラフィック座標を指定します。

座標は、通常は表示する文字列の左上隅の座標を指定しますが、回転角度が 90 度、180 度、270 度のときはそれぞれ、左下、右下、右上隅の座標を指定しなければなりません。

mag\_x、mag\_y には、文字の拡大倍率を指定します。

mag\_x は横の倍率、mag\_y は縦の倍率ですが、これは文字の縦横に対しての指定で、画面の縦横に対してではないことに注意してください。

color には、文字の色を指定します。

font\_type には次の文字フォントのタイプを指定します。

- 0 : 12×12 ドットフォント
- 1 : 16×16 ドットフォント
- 2 : 24×24 ドットフォント

angle には次の角度コードを指定します。

- 0 : 通常
- 1 : 90 度回転
- 2 : 180 度回転
- 3 : 270 度回転

**戻 り 値**

終了コードを返します。

- 0 : 正常終了
- 1 : グラフィックは使用不可

# TCOLOR

レベル 0

**書 式** #include <ioclib.h>

```
void TCOLOR (TEXTCOLOR) ;  
int TEXTCOLOR; /* テキストのカラー */
```

**機 能** テキストのカラーを指定します。

TEXTGET, TEXTPUT, CLIPPUT などの関数を使用するテキスト画面を決めます。

TEXTCOLOR には 0~15 まで指定できますが、原則として 1、2、4、8 のみ指定してください。

- 1 : 0xE00000~0xE1FFFF の面を使用
- 2 : 0xE20000~0xE3FFFF の面を使用
- 4 : 0xE40000~0xE5FFFF の面を使用
- 8 : 0xE60000~0xE7FFFF の面を使用

**戻 り 値** 戻り値はありません。

# TEXTGET

レベル 0

**書 式** #include <iocslib. h>

```
void TEXTGET (XDOT, YDOT, BUF);
int XDOT; /* Xドット座標 */
int YDOT; /* Yドット座標 */
struct FNTBUF *BUF; /* データバッファアドレス */
struct FNTBUF {
    short xl; /* X方向のドット数 */
    short yl; /* Y方向のドット数 */
    unsigned char buffer [72]; /* データバッファ */
};
```

**機 能** XDOT および YDOT で指定されたドット座標から、BUF で指定されたアドレスへパターンを読み込みます。

BUF にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないように注意してください。

BUF には、パターンの X 方向のドット数、Y 方向のドット数をそれぞれ short で指定してこの関数を呼びます。

buffer は読み込んだパターンが入るだけの十分な大きさが必要です。

構造体 FNTBUF のデータサイズを越える場合には、メモリ確保用の関数 malloc などて確保した領域を使用してください。

**戻 り 値** 戻り値はありません。

# TEXTPUT

レベル 0

**書 式** #include <iocslib. h>

```
void TEXTPUT (XDOT, YDOT, BUF) ;
int XDOT; /* Xドット座標 */
int YDOT; /* Yドット座標 */
struct FNTBUF *BUF; /* データバッファアドレス */
struct FNTBUF {
    short xl; /* X方向のドット数 */
    short yl; /* Y方向のドット数 */
    unsigned char buffer [72]; /* データバッファ */
};
```

**機 能** BUFで指定されたアドレスから、XDOTおよびYDOTで指定されたドット座標へ、パターンを書き出します。

BUFにはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

BUFには、パターンのX方向のドット数、Y方向のドット数、およびパターンデータを格納してください。

bufferには、指定したX、Y方向のドット数に必要なだけのデータを格納しておいてください。

構造体FNTBUFのデータサイズを越える場合には、メモリ確保用の関数mallocなどで確保した領域を使用してください。

**戻り値** 戻り値はありません。

# TGUSEMD

## レベル 0

**書 式** #include <ioclib. h>

```
int TGUSEMD (TEXT_GR, MODE) ;
int TEXT_GR; /* 画面の種類 */
int MODE; /* 画面のモード */
```

**機 能** TEXT\_GR で指定した画面のモードの設定、または読みとりを行います。  
TEXT\_GR、MODE に指定できる内容は次の通りです。

● TEXT\_GR

0 : グラフィック (0xC00000~0xDFFFFFF)

1 : テキスト (0xE40000~0xE7FFFF)

● MODE

0 : 誰も使用していない

1 : システムで使用している

(ソフトキー/電卓/RAM ディスク)

2 : アプリケーションで使用している

3 : アプリケーションで使用して、壊れたままである

-1 : 画面のモードを読みとる

ただし TEXT\_GR に 1 を指定した場合、MODE に 0 または 3 を指定すると  
MODE に 1 を指定したことになります。

**戻 り 値** 画面のモードを返します。



プログラム例

```

#include <stdio.h>
#include <iocslib.h>

#define M_GRP 0 /* グラフィックモード */
#define M_TEXT 1 /* テキストモード */
#define S_NOT 1 /* 誰も使用していない */
#define S_SYS 2 /* システムが使用 */
#define S_APP 3 /* アプリケーションが使用 */
#define S_APPD 4 /* 壊れたまま */
#define S_READ (-1) /* 画面モード読み取り */

main()
{
    int r;

    if ((r = TGUSEMD(M_TEXT, S_READ)) == S_NOT)
        puts("使用されていません。");
    else if (r == S_SYS)
        puts("システムで使用。");
    else
        puts("アプリケーションで使用。");
}

```

女 香

機 器

機 器

# TIMEASC

レベル 0

**書 式** #include <iocslib.h>

```
int TIMEASC (BINTIME, BUF);
int BINTIME; /* 時刻 (2進数) */
unsigned char *BUF; /* 文字列格納領域へのポインタ */
```

**機 能** 2進データを、時刻を表す文字列に変換し、格納します。  
BINTIME の形式と指定内容は次の通りです。

```
00000000 hhhhhhhh mmmmmmmm ssssssss
```

```
hhhhhhh : 時 (00~23)
mmmmmmm : 分 (00~59)
sssssss : 秒 (00~59)
```

BUF には、時刻を表す文字列の格納領域の先頭アドレスを指定します。

この領域は、最低9バイト以上確保してください。

BUF にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

BUF には、次のような形式で格納されます。

```
"12:05:30"
```

**戻り値** エラーが発生したときは-1を返します。

## プログラム例

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

main()
{
    char str[9]; /* 文字列格納領域 */
    int bin; /* 時刻(2進数) */

    bin = TIMEBIN(TIMEGET());

    TIMEASC(bin, str);

    printf("%s\n", str);
}
```

# TIMEBCD

レベル 0

**書 式** #include <iocslib.h>

```
int TIMEBCD (BINTIME) ;
int BINTIME ; /* 時刻 (2進数) */
```

**機 能** 2進数の時刻を、時計に設定できる形式に変換します。

BINTIME の形式と指定内容は次の通りです。

```
00000000 hhhhhhhh mmmmmmmm ssssssss
```

hhhhhhh : 時 (00~23)

mmmmmmm : 分 (00~59)

sssssss : 秒 (00~59)

上記の時・分・秒はすべて 2 進数です。

**戻 り 値** 次の内容を返します。

```
0000TTTT HHHHHHHH MMMMMMMM SSSSSSS
```

TTTT : 1 (24 時間計であることを示します)

HHHHHHH : 時 (00~23)

MMMMMMM : 分 (00~59)

SSSSSSS : 秒 (00~59)

上記の時、分、秒はすべて BCD2 桁です。

エラーが発生したときは -1 を返します。

```
#include <iocslib.h>
#include <ioclib.h>

main()
{
    char str[10]; /* 文字列領域 */
    int pin; /* 時刻 (2進数) */
    pin = TIMEBIN(TIMEBCD(1));
    TIMEASC(pin, str);
    printf("%s\n", str);
}
```

## プログラム例

```

#include <iocslib.h>

#define HOU 10
#define MIN 28
#define SEC 40

main()
{
    int bin; /* 時刻(2進数) */
    int bcd; /* 時刻(BCD) */

    bin = (HOU << 16) | (MIN << 8) | (SEC << 0);
    bcd = TIMEBCD(bin);

    TIMESET(bcd); /* 時刻を時計に設定 */
}

```

# TIMEBIN

レベル 0

**書 式** #include <iocslib.h>

```
int TIMEBIN (BCDTIME) ;
int BCDTIME ; /* 時刻 (BCD) */
```

**機 能** 時刻を BCD から 2 進数に変換します。

BCDTIME の形式と指定内容は次の通りです。

```
00000000 HHHHHHHH MMMMMMMM SSSSSSS
```

HHHHHHHH : 時 (00~23)

MMMMMMMM : 分 (00~59)

SSSSSSSS : 秒 (00~59)

上記の時・分・秒はすべて BCD2 桁です。

**戻 り 値** 次の内容を返します。

```
00000000 hhhhhhhh mmmmmmmm sssssss
```

hhhhhhh : 時 (00~23)

mmmmmmm : 分 (00~59)

sssssss : 秒 (00~59)

上記の時, 分, 秒はすべて 2 進数です。

## プログラム例

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

main()
{
    int bcdtime; /* 時刻(BCD) */
    int bintime; /* 時刻(2進数) */

    bcdtime = TIMEGET();

    bintime = TIMEBIN(bcdtime);
    printf("%d時", (bintime >> 16) & 0xff);
    printf("%d分", (bintime >> 8) & 0xff);
    printf("%d秒", (bintime >> 0) & 0xff);
    printf("\n");
}
```

# TIMECNV

レベル 0

**書 式** #include <iocslib.h>

```
int TIMECNV (ADDRESS) ;
unsigned char *ADDRESS ; /* 時刻を示す文字列へのポインタ */
```

**機 能** 時刻を表す文字列を2進データに変換します。

ADDRESS には、時刻を表す文字列の先頭アドレスを指定します。

ADDRESS にはスーパーバイザ領域のアドレスも指定できますが、誤ったアドレスを指定しないよう注意してください。

また、時刻を表す文字列は次のように指定します。

"12:05:30"

区切り記号は:(コロン)以外でもかまいません。

**戻 り 値** 2進数に変換した時刻を返します。

時刻の形式は次の通りです。

00000000 hhhhhhhh mmmmmmmm ssssssss

hhhhhhh : 時 (00~23)

mmmmmmm : 分 (00~59)

sssssss : 秒 (00~59)

エラーが発生したときは-1を返します。

## プログラム例

```
#include <iocslib.h>

main()
{
    int bcd; /* 時刻(BCD) */
    char *str;

    str = "10:08:40";

    bcd = TIMEBCD(TIMECNV(str));

    TIMESET(bcd);
}
```

# TIMEGET

レベル 0

**書 式** #include <ioclib.h>

```
int TIMEGET ( );
```

**機 能** 時計から時刻を読み込みます。

**戻 り 値** 次の内容を返します。

```
00000000 HHHHHHHH MMMMMMMM SSSSSSS
```

HHHHHHHH : 時 (00~23)

MMMMMMMM : 分 (00~59)

SSSSSSSS : 秒 (00~59)

上記の時, 分, 秒はすべて BCD2 桁です。

## プログラム例

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

main()
{
    int bcd; /* 時刻(2進数) */
    int bin; /* 時刻(BCD) */

    bcd = TIMEGET();
    bin = TIMEBIN(bcd);

    printf("%d時", (bin >> 16) & 0xff);
    printf("%d分", (bin >> 8) & 0xff);
    printf("%d秒", (bin >> 0) & 0xff);
    printf("%n");
}
```

# TIME\_PR

## レベル 1

**書 式** # include <doslib. h>

```
long TIME_PR ( ) ;
```

**機 能**

現在のタイマーのカウンタ値（ミリ秒単位）を返します。バックグラウンドで複数のスレッドが並行して動作している場合、プログラムで一定の時間を計るためには、どのスレッドが動作していても、一定して変化するカウンタが必要になります。

longでの最大値を越えると0に戻るのので、そのまま前回に返されたタイマーのカウンタ値との引き算で経過時間が分かります。

**戻 り 値**

現在のタイマーのカウンタ値を返します。

**参 照 関 数**

OPEN\_PR, KILL\_PR, GET\_PR, SUSPEND\_PR, SLEEP\_PR, SEND\_PR, CHANGE\_PR

# TIMERDST

レベル 0

**書 式** #include <iocslib.h>

```
int TIMERDST (ADDRESS, MODE, COUNTER);  
unsigned char *ADDRESS; /* 割り込み処理アドレス */  
int MODE; /* モード */  
int COUNTER; /* カウンタ */
```

**機 能** MFP の TIMER-D による割り込みの設定を行います。

ADDRESS には割り込み処理アドレスを指定します (0 ならば割り込み禁止)。

MODE には次の内容を指定します。

- 1 : 1 $\mu$ sec 単位
- 2 : 2.5 $\mu$ sec 単位
- 3 : 4 $\mu$ sec 単位
- 4 : 12.5 $\mu$ sec 単位
- 5 : 16 $\mu$ sec 単位
- 6 : 25 $\mu$ sec 単位
- 7 : 50 $\mu$ sec 単位

COUNTER には 0 から 255 までの値を指定します。

MODE の値に COUNTER の値を乗じた値の単位で、割り込みが行われます。

なお、Human68k ver. 2.0 の動作中は、割り込みを変更できません。

**戻 り 値** 割り込みが可能なときは 0、すでに使用中の場合には 0 以外の値を返します。

# TIMESET

レベル 0

**書式** #include <iocslib.h>

```
void TIMESET (BCDTIME) ;
int BCDTIME; /* 時刻 (BCD) */
```

**機能** 時計に時刻を設定します。

BCDTIME の形式と指定内容は次の通りです。

```
0000TTTT HHHHHHHH MMMMMMMM SSSSSSS
```

TTTT : 1:24 時間計

HHHHHHHH : 時 (00~23)

MMMMMMMM : 分 (00~59)

SSSSSSSS : 秒 (00~59)

上記の時・分・秒はすべて BCD2 桁です。

**戻り値** 戻り値はありません。

## プログラム例

```
#include <iocslib.h>

#define HOU 10
#define MIN 8
#define SEC 40
#define CLOCK24 0x1000000
#define CLOCK12 0x0000000

main()
{
    int bin; /* 時刻(2進数) */
    int bcd; /* 時刻(BCD) */

    bin = CLOCK24 | (HOU << 16) | (MIN << 8) | (SEC << 0);

    bcd = TIMEBCD(bin);

    TIMESET(bcd); /* 時刻を時計に設定 */
}
```

# TPALET

レベル 0

**書 式** #include <iocslib. h>

```
int TPALET (PALETNO, COLORCODE) ;
int PALETNO; /* パレット番号 */
int COLORCODE; /* カラーコード */
```

## 機 能

テキストのパレットを設定します。

PALETNO はテキストパレットの番号です。0~15 を指定します。

テキストパレットの 0~3 は、テキストに表示する文字の色指定に使われます。

テキストパレットが 4~7、8~15 の場合にはそれぞれ 4~7、8~15 に同一のカラーコードが設定されます。

4~7、8~15 は、ソフトキーボード/電卓で使用されます。

0~3 : テキストカラー 0~3

4~7 : ソフトキーボード/電卓カラー

(4~7 に同じカラーコードをセット)

8~15 : ソフトキーボード/ 電卓カラー

(8~15 に同じカラーコードをセット)

COLORCODE には 0~65535 を指定します。

なお、COLORCODE に -1 を設定するとパレットの読み出しのみ行い、-2 ならばパレットをシステムの既定値にします。

## 戻 り 値

COLORCODE が -1 のときには、現在のカラーコードを返します。

# TPALET2

レベル 0

**書 式** #include <iocslib.h>

```
int TPALET2 (PALETNO, COLORCODE);
int PALETNO; /* パレット番号 */
int COLORCODE; /* カラーコード */
```

**機 能** テキストのパレットを設定します。

PALETNO には、0~15 を指定します。

COLORCODE には、0~65535 を指定します。

なお、COLORCODE に -1 を設定するとパレットの読み出しのみ行い、-2 ならばパレットをシステムの既定値に戻します。

**戻 り 値** COLORCODE が -1 のときは、現在のカラーコードを返します。

## プログラム例

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

main()
{
    int pno; /* パレット番号 */
    int color; /* カラーコード */

    for (pno = 0; pno < 16; pno++) {
        /* パレット読出し */
        color = TPALET2(pno, -1);
        printf("palet No.%d の ", pno);
        printf("ColorCode: 0x%x\n", color);
    }
}
```

# TRAP15

レベル 0

**書 式** # include <iocslib.h>

```
int TRAP15 (inregs, outregs);  
struct REGS { /* レジスタを直接アクセスするための構造体 */  
    int d0;  
    int d1;  
    int d2;  
    int d3;  
    int d4;  
    int d5;  
    int d6;  
    int d7;  
    int a1;  
    int a2;  
    int a3;  
    int a4;  
    int a5;  
    int a6;  
} *inregs; /* 呼び出しプログラムに渡すレジスタの値を格納した構造体  
           へのポインタ */  
struct REGS *outregs; /* 呼び出しプログラムから返されるレジスタの値  
                       が格納される構造体へのポインタ */
```

**機 能**

inregs で指定した構造体中のデータを各レジスタに書き込んだ後、直接 trap # 15 命令を実行します。

そして、実行後のレジスタの値を outregs で指定された構造体へ格納します。引数であるレジスタの初期値は自分で確保した構造体に格納し、さらに TRAP15 にはその構造体のポインタを渡します。

trap # 15 命令は通常 IOCS コールを実行するので、inregs->d0 に IOCS コール番号を書き込み、他のメンバに必要な値を書き込んだ後で、TRAP15 関数を実行することで直接 IOCS コールを行うことができます。

inregs に格納した値がおかしいと暴走する可能性があるので十分注意して使用してください。

**戻 り 値**

trap # 15 命令により呼び出された IOCS コールが返す d0.1 の値を返します。

## プログラム例

```

#include      <stdio.h>
#include      <iocslib.h>

main()
{
    struct REGS  inregs;
    struct REGS  outregs;
    int          r;

    inregs.d0 = 0x01;
    r = TRAP15(&inregs, &outregs);
    if (r == 0)
        printf("キーが押されていない\n");
    else {
        printf("スキャンコード0x%x\n", (r >> 8) & 0xff);
        printf("内部コード      0x%x\n", r & 0xff);
    }
}

```

# TVCTRL

## レベル 0

**書 式** #include <iocslib. h>

```
void TVCTRL (CODE) ;
int CODE; /* TV コントロールコード */
```

**機 能** CODEにTVコントロールコード(6ビット)を指定し、TVの操作を行います。  
TV コントロールコードは次の通りです。

- 0x01 : ボリュームを上げる
- 0x02 : ボリュームを下げる
- 0x03 : ボリュームを標準にする
- 0x04 : チャンネルコール
- 0x05 : テレビ画面 (初期化・リセット)
- 0x06 : 音声ミュート
- 0x07 : 電源 ON
- 0x08 : テレビ/コンピュータ
- 0x09 : テレビ/外部、コンピュータノーマル/オーバー
- 0x0A : コントラスト標準
- 0x0B : チャンネルアップ
- 0x0C : チャンネルダウン
- 0x0D : 電源 OFF
- 0x0E : 電源 ON/OFF
- 0x0F : スーパー 1
- 0x10 : チャンネル 1
- 0x11 : チャンネル 2
- 0x12 : チャンネル 3
- 0x13 : チャンネル 4
- 0x14 : チャンネル 5
- 0x15 : チャンネル 6
- 0x16 : チャンネル 7
- 0x17 : チャンネル 8
- 0x18 : チャンネル 9
- 0x19 : チャンネル 10
- 0x1A : チャンネル 11
- 0x1B : チャンネル 12
- 0x1C : テレビ画面 (0x05)
- 0x1D : コンピュータ画面 (0x05+0x08)

TXBOX

0x1E

0x1E : スーパー 1 (0x05+0x0F)

0x1F : スーパー 2 (0x05+0x0F+0x0A)

0x21~0x3F : 電源を入れた後、上記のファンクションを実行する  
(0x20+0x01~0x1F)\*

**戻り値** 戻り値はありません。

**プログラム例**

```
#include <iocslib.h>

main()
{
    int code; /* TVコントロールコード */
    int i;
    int j;

    code = 0x08; /* テレビ/コンピュータ */
    TVCTRL(code);

    code = 0x0b; /* チャンネルアップ */
    for (i = 1; i < 13; i++) {
        for (j = 0; j < 0x7fff; j++)
            ;
        TVCTRL(code);
    }
    code = 0x1d; /* コンピュータ画面 */
    TVCTRL(code);
}
```

# TXBOX

レベル 0

**書 式**

```
#include <iocslib.h>

void TXBOX (tboxptr) ;
struct TBOXPTR *tboxptr;
struct TBOXPTR {
    unsigned short vram_page; /* テキストのページ */
    short x; /* 始点 X 座標 */
    short y; /* 始点 Y 座標 */
    short x1; /* 最終点までの X の長さ */
    short y1; /* 最終点までの Y の長さ */
    unsigned short line_style; /* ラインスタイル */
};
```

**機 能**

テキスト画面に四角形を描きます。

**戻り値**

戻り値はありません。

# TXFILL

レベル 0

**書 式** #include <iocslib. h>

```
void TXFILL (txfillptr);  
struct TXFILLPTR *txfillptr;  
struct TXFILLPTR {  
    unsigned short vram_page; /* テキストのページ */  
    short x; /* 始点 X 座標 */  
    short y; /* 始点 Y 座標 */  
    short x1; /* 最終点までの X の長さ */  
    short y1; /* 最終点までの Y の長さ */  
    unsigned short fill_patn; /* ぬりつぶしのパターン */  
};
```

**機 能** テキスト画面にぬりつぶした四角形を描きます。

**戻 り 値** 戻り値はありません。

# TXRASCOPY

レベル 0

**書 式** #include <iocslib. h>

```
void TXRASCOPY (SOR_DEST, COPY, MODE) ;  
int SOR_DEST ;  
int COPY ; /* コピーする個数 */  
int MODE ; /* コピーする画面 */
```

**機 能**

テキスト画面のラスタコピーを行います。  
SOR\_DEST には、ソース×256+デスティネーションを指定します。  
各8ビットは4ラスタ単位の値となります。  
COPY にはコピーする個数を指定します (4ラスタを1個とします)。  
MODE にはコピーする画面と方向を指定します。  
画面は同時に4面まで可能です。

ビット 0 : テキスト 0  
ビット 1 : テキスト 1  
ビット 2 : テキスト 2  
ビット 3 : テキスト 3  
ビット 15 : 0 = インクリメント  
          1 = デクリメント

**戻 り 値** 戻り値はありません。

# TXREV

レベル 0

**書 式** #include <iocslib. h>

```
void TXREV (trevptr) ;  
struct TREVPtr *trevptr ;  
struct TREVPtr {  
    unsigned short vram_page ; /* テキストのページ */  
    short x ; /* 始点の X 座標 */  
    short y ; /* 始点の Y 座標 */  
    short x1 ; /* 最終点までの X の長さ */  
    short y1 ; /* 最終点までの Y の長さ */  
};
```

**機 能** テキスト画面のリバースを行います。

**戻り値** 戻り値はありません。

# TXXLINE

レベル 0

**書 式** #include <iocslib. h>

```
void TXXLINE (TXXPTR) ;
struct XLINEPTR *TXXPTR;
struct XLINEPTR {
    unsigned short vram_page; /* テキストのページ */
    short x; /* 始点 X 座標 */
    short y; /* 始点 Y 座標 */
    short x1; /* 終点までの X の長さ */
    unsigned short line_style; /* ラインスタイル */
};
```

**機 能** テキスト画面に水平方向のラインを描きます。

**戻 り 値** 戻り値はありません。

# TXYLNE

レベル 0

**書 式** #include <ioclib.h>

```
void TXYLNE (TXYPTR) ;
struct YLINERTR *TXYPTR;
struct YLINERTR {
    unsigned short vram_page; /* テキストのページ */
    short x; /* 始点 X 座標 */
    short y; /* 始点 Y 座標 */
    short yl; /* 終点までの Y の長さ */
    unsigned short line_style; /* ラインスタイル */
};
```

**機 能** テキスト画面に垂直方向のラインを描きます。

**戻 り 値** 戻り値はありません。

```

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <ioclib.h>
#include <doslib.h>

#define LINE_SIZE 256
static char buf[LINE_SIZE];

int main()
{
    int fd;
    if ((fd = open("A:\\vjoy.dat", O_WRONLY | O_CREAT, 0666)) < 0)
        puts("can't open");
    else if (lock(fd, 0, LINE_SIZE) < 0)
        puts("can't lock");
    else if (read(fd, buf, LINE_SIZE) < 0)
        puts("can't read");
    if (unlock(fd, 0, LINE_SIZE) < 0)
        puts("can't unlock");
    close(fd);
}

```

# UNLOCK

レベル 1

**書式** #include <doslib.h>

```
int UNLOCK (fileno, offset, len) ;
int fileno; /* ファイルハンドル */
int offset; /* ロック解除する部分へのオフセット */
int len; /* ロック解除する部分の長さ */
```

**機能**

ファイルのロック解除を行い、他のプロセスからのファイルアクセスを許可します。

fileno は、ロック解除したいファイルのファイルハンドルを指定します。

offset には、ファイル中のロック解除したい部分の、ファイルの先頭からのオフセットを指定します。

len には、ファイル中のロック解除したい部分のバイト数を指定します。

この UNLOCK 関数と LOCK 関数により、ファイルアクセスの排他制御を行い、自分以外のプロセスが勝手にファイルを書き変えるのを防ぐことができます。

**戻り値**

負の数ならエラーを表します。

**参照関数**

LOCK

**プログラム例**

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

#define SIZE 256

static char buf[SIZE];

main()
{
    int fd;

    if ((fd = open("A:¥¥vjob.dic", O_RDONLY|O_BINARY)) == -1)
        puts("can't open");
    else if (LOCK(fd, 0, SIZE) < 0)
        puts("can't lock");
    else if (read(fd, buf, SIZE) != SIZE)
        puts("can't read");
    if (UNLOCK(fd, 0, SIZE) < 0)
        puts("can't unlock");
    close(fd);
}
```

# VDISPST

レベル 0

**書 式** #include <iocslib. h>

```
int VDISPST (ADDRESS, MODE, COUNTER) ;
unsigned char *ADDRESS; /* 割り込み処理アドレス */
int MODE; /* モード */
int COUNTER; /* カウンタ */
```

**機 能**

V-DISP による割り込みを、指定カウントごとに行います。

ADDRESS には割り込み処理アドレスを指定します (0 ならば割り込み禁止)。

MODE には次の内容を指定します。

- 0 : 垂直帰線期間をカウント
- 1 : 垂直表示期間をカウント

COUNTER には 0 から 255 までの値を指定します。

**戻 り 値**

割り込み可能なときは 0、すでに使用中の場合には 0 以外の値を返します。

# VERIFY

レベル 1

**書式** #include <doslib.h>

```
void VERIFY (flag) ;  
int flag; /* ベリファイフラグの設定モード */
```

**機能** ベリファイフラグの設定を行います。  
flag に設定できる内容は次の通りです。

- 0 : ベリファイしない
- 1 : ベリファイする

**戻り値** 戻り値はありません。

## プログラム例

```
#include <stdio.h>  
#include <doslib.h>  
  
#define ON 1  
#define OFF 0  
  
main(int argc, char *argv[])  
{  
    if (argc == 1) {  
        printf("verify は %s です。%n",  
              (VERIFYG() ? "on" : "off"));  
    }  
    else if (strcmp(argv[1], "on") == 0) {  
        VERIFY(ON);  
        printf("verify を on にしました。%n");  
    }  
    else if (strcmp(argv[1], "off") == 0) {  
        VERIFY(OFF);  
        printf("verify を off にしました。%n");  
    }  
    else  
        printf("Usage: verify [on|off]%n");  
}
```

# VERIFYG

## レベル 1

**書式** #include <doslib.h>

```
int VERIFYG ( );
```

**機能** ベリファイフラグの設定状況を調べます。

**戻り値** 設定状況を示す次の値を返します。

0 : ベリファイは行わない

1 : ベリファイを行う

### プログラム例

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

#define ON 1
#define OFF 0

main(int argc, char *argv[])
{
    if (argc == 1) {
        printf("verify は %s です。%n",
            (VERIFYG() ? "on" : "off"));
    }
    else if (strcmp(argv[1], "on") == 0) {
        VERIFY(ON);
        printf("verify を on にしました。%n");
    }
    else if (strcmp(argv[1], "off") == 0) {
        VERIFY(OFF);
        printf("verify を off にしました。%n");
    }
    else
        printf("Usage: verify [on|off]%n");
}
```

# VERNUM

レベル 1

**書 式** #include <doslib.h>

```
int VERNUM ( );
```

**機 能** Human68k のバージョン番号を調べます。

**戻 り 値** バージョン番号を返します。

バージョン番号の形式は次の通りです。

上位 16 ビット = '68'

下位 16 ビット = 整数部×256+小数部

## プログラム例

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

main()
{
    int ver; /* バージョン番号 */

    ver = VERNUM();
    printf("c%c ", ver >> 24, ver >> 16);
    printf("version ");
    printf("%d.%d\n", (ver >> 8) & 0xff, ver & 0xff);
}
```

# VPAGE

レベル 0

**書 式** #include <iocslib.h>

```
int VPAGE (MODE) ;  
int MODE; /* 表示ページの ON/OFF 指定 */
```

**機 能** グラフィック画面の表示ページを設定します。  
MODE は表示ページの ON/OFF を指定します。

ビット 0 : 0 = 0 ページは表示しない  
          1 = 0 ページを表示する  
ビット 1 : 0 = 1 ページは表示しない  
          1 = 1 ページを表示する  
ビット 2 : 0 = 2 ページは表示しない  
          1 = 2 ページを表示する  
ビット 3 : 0 = 3 ページは表示しない  
          1 = 3 ページを表示する

**戻 り 値** 終了コードを返します。

- 0 : 正常終了
- 1 : グラフィックは使用不可
- 2 : ページ引数が指定どおりになっていない
- 3 : 指定されたページは現在のモードでは設定不可

# WAIT

## レベル 1

**書 式** #include <doslib.h>

int WAIT ( ) ;

**機 能** 自分が直前に実行した子プロセスの終了コードを返します。

**戻 り 値** 終了コードを返します。

### プログラム例

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

main()
{
    int end_stas; /* 終了ステータス */
    end_stas = WAIT();
    exit(end_stas);
}
```

# WINDOW

レベル 0

**書 式** #include <iocslib.h>

```
int WINDOW (SX, SY, EX, EY) ;
int SX; /* ウィンドウ左上の X 座標 */
int SY; /* ウィンドウ左上の Y 座標 */
int EX; /* ウィンドウ右下の X 座標 */
int EY; /* ウィンドウ右下の Y 座標 */
```

**機 能**

グラフィック画面のウィンドウを設定します。  
座標を指定するときは、 $SX \leq EX$ 、 $SY \leq EY$  となっていなければなりません。  
また、指定できるのはいずれも 0~1023 までです。

**戻り値**

終了コードを返します。

- 0 : 正常終了
- 1 : グラフィックは使用不可
- 2 : 引数が指定通りになっていない

# WIPE

レベル 0

**書 式** #include <iocslib. h>

int WIPE ( ) ;

**機 能** グラフィック画面をクリアします。

**戻り値** 終了コードを返します。

0 : 正常終了

-1 : グラフィックは使用不可

# WRITE

## レベル 1

**書式** #include <doslib. h>

```
int WRITE (fileno, buffer, size) ;  
int fileno; /* 該当ファイルのファイルハンドル */  
unsigned char *buffer; /* 書き出しデータ格納領域へのポインタ */  
int size; /* 書き出しバイト数 */
```

**機能** buffer が指す領域からデータを、fileno で指定するファイルハンドルへ、size で指定するバイト数分、書き出します。

**戻り値** 書き出したバイト数を返します。もし、指定した size より小さい値や、0 が返された場合はディスクフルが発生した事を表します。  
したがって、ディスクフルでは負の数ではなく、正の数 (0 も) が返されることに注意して下さい。  
負の数ならエラーを表します。

**参照関数** READ



# 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

# 疑 問

## のりててトローサーエ 式りお

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。

書宣務長ののりててト  
くべり

## 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。

## 2 コンパイル

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、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.l` です。ユーザーが自身でライブラリを作成した場合、`clib.l` 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 `l`）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、「アセンブラマニュアル」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、リンク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、およびリンク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 
```

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。



# 索引 50音順

朗音 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
アポート処理	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
引数配列	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、81
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 .....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
スプライトプレーンデータ	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
バックグラウンドテキスト	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
ブレーク値	31
プロセス	53
プロセス制御関数	18、23、53
分数部	52

## フ

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
------	-------

**ロ**

論理エラー .....19, 92  
 ロングワードデータ .....30

**ワ**

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

**D**

DEFALT .....12  
 define .....12  
 =dcm\$IAIRU .....14  
 direct .....17, 30, 32  
 DMA 転送 .....32  
 DMA 転送制御 .....32  
 DOS コマンド .....32  
 double .....17, 30, 32  
 double .....32  
 div .....12  
 div .....12

**E**

EOF .....12, 31  
 end .....12  
 EOS .....12  
 error .....17, 30  
 exception 構文 .....12  
 exit .....12  
 EXIT\_FAILURE .....12  
 EXIT\_SUCCESS .....12

**A**

ADPCM .....17, 30, 32  
 ADPCM 制御 .....17, 30, 32  
 ANK .....12  
 ANSI 規格 .....12  
 AR .....12  
 ASCII .....12  
 ASCII .....12  
 assert .....12  
 assert .....12  
 assert .....12  
 audio .....12  
 AUX .....12

**B**

BASIC ライブラリ .....12  
 basic .....17, 19, 32  
 basic .....17, 19, 32  
 BASTOC .....12  
 BCD 表現 .....32  
 BC コントロールマスク .....80  
 BC マスク .....80  
 BG .....80  
 BG .....80  
 BOOLEAN .....12  
 BUSIZ .....12, 34  
 BUF\_SIZ .....12  
 BYTE .....12

**C**

C 言語の識別子 .....32  
 C ライブラリ .....12  
 char .....12  
 class .....12, 17, 19  
 clear .....12

# 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
cleareerr 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 <fctype. 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**

jcype. 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  
 longjmp 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、78  
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  
tell .....48  
time 関数 .....12、18  
timeb 構造体 .....16、59

### T

timeb.h .....16、18、26、59

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

void 型 .....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_strchr .....	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
<b>C</b> .....		clock .....	194
cabs .....	179	close .....	195
calloc .....	180	CLOSE .....	117
C_CLS_AL .....	94	clrerr .....	193
C_CLS_ED .....	95	cls .....	196
C_CLS_ST .....	96	color .....	197

COMINP .....118  
 COMMON\_CK .....119  
 COMMON\_DEL .....120  
 COMMON\_FRE .....121  
 COMMON\_LK .....123  
 COMMON\_RD .....125  
 COMMON\_WT .....127  
 COMOUT .....129  
 CONSNS .....130  
 console .....198  
 contrast .....199  
 CONTRAST .....131  
 cos .....200  
 cosh .....200  
 COUTSNS .....132  
 C\_PRINT .....133  
 cprintf .....201  
 C\_PUTC .....134  
 cputs .....202  
 creat .....203  
 CREATE .....135  
 C\_RIGHT .....136  
 crt .....204  
 CRTCRAS .....137  
 CRTMOD .....138  
 cscanf .....205  
 csrlin .....206  
 ctime .....207  
 C\_UP .....139  
 C\_UP\_S .....140  
 CURDIR .....141  
 CURDRV .....142  
 C\_WIDTH .....143  
 C\_WINDOW .....144

**D**

DAKJOB .....145  
 DATEASC .....146  
 DATEBIN .....148  
 DATECNV .....150  
 DAYASC .....151  
 DEFCHR .....152  
 DELETE .....153  
 DENSNS .....154  
 difftime .....208  
 DISKRED .....155  
 DISKRED2 .....157  
 DISKWRT .....159  
 DISKWRT2 .....161  
 div .....209  
 DMAMODE .....163  
 DMAMOV\_A .....164  
 DMAMOVE .....166  
 DMAMOV\_L .....168  
 dqsort .....210  
 DRVCTRL .....170  
 DRVXCHG .....172  
 dskf .....211  
 DSKFRE .....173  
 dtasc .....212  
 dtcnv .....214  
 dup .....215  
 DUP .....174  
 DUP0 .....175  
 dup2 .....215  
 DUP2 .....176

**E**

ecvt .....217  
 eof .....219

## 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	islower .....	315
INP232C .....	238	ISNS232C .....	257
INPOUT .....	239	isodigit .....	315
instr .....	313	ispnkana .....	314
INTVCG .....	240	isprint .....	315
INTVCS .....	241	isprkana .....	314
IOCTRLDVCTL .....	242	ispunct .....	315
IOCTRLDVGT .....	243	isspace .....	315
IOCTRLFDCTL .....	244	isupper .....	315
IOCTRLFDGT .....	245	isxdigit .....	315
IOCTRLGT .....	246	itoa .....	320
IOCTRLIS .....	248		
IOCTRLOS .....	249	<b>J</b>	
IOCTRLRD .....	250	jisalpha .....	321
IOCTRLRH .....	251	jisdigit .....	321
IOCTRLRTSET .....	252	jishira .....	321
IOCTRLST .....	253	jiskata .....	321
IOCTRLWD .....	254	jiskigou .....	321
IOCTRLWH .....	255	jisl0 .....	322
IPLERR .....	256	jisl1 .....	322
isalkana .....	314	jisl2 .....	322
isalnmkana .....	314	jislower .....	321
isalnum .....	315	JISSFT .....	258
isalpha .....	315	jisspace .....	321
isascii .....	315	jisupper .....	321
isatty .....	317	jiszen .....	322
iscntrl .....	315	JOYGET .....	259
iscsym .....	315	jstrchr .....	323
iscsymf .....	315	jstrcmp .....	324
isdigit .....	315	jstrncmp .....	325
isgraph .....	315	jstrrchr .....	326
isgrkana .....	314		
iskana .....	318	<b>K</b>	
iskanji .....	319	kbhit .....	327
iskanji2 .....	319	KEEPPR .....	260
iskmoji .....	318		
iskpun .....	318		

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

PSET .....331  
PSPSET .....332  
put .....412  
putc .....414  
putch .....416  
putchar .....414  
PUTCHAR .....334  
putenv .....417  
PUTGRM .....335  
putl .....419  
puts .....420  
putw .....421  
**Q**  
qsort .....422  
**R**  
raise .....424  
rand .....426  
randomize .....427  
RASSIGN .....336  
rbrk .....428  
read .....429  
READ .....337  
realloc .....431  
remove .....547  
rename .....433  
RENAME .....338  
repmem .....435  
rewind .....436  
rgb .....437  
rlsmem .....291  
rlsml .....291  
RMACNV .....339  
rmdir .....438

RMDIR .....340  
rmtmp .....439  
rmtmp .....439  
rnd .....440  
ROMVER .....341  
rstmem .....441  
**S**  
sbrk .....442  
scanf .....443  
screen .....447  
SCROLL .....342  
SEEK .....343  
SEND\_PR .....344  
SET232C .....346  
SETBLOCK .....348  
setbuf .....449  
SETDATE .....349  
SETENV .....350  
setjmp .....451  
setmem .....367  
setmode .....453  
setmspos .....455  
setnbf .....456  
SETPDB .....351  
SETTIM2 .....352  
SETTIME .....353  
setvbuf .....456  
SFTJIS .....354  
sgn .....457  
signal .....458  
sin .....461  
sinh .....461  
sizmem .....462  
SKEY\_MOD .....355  
SKEYSET .....357

## 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	strcspn	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	strmfh	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	strev	506
sprintf	476	strset	507
S_PROCESS	371	strsfh	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

strupr .....519  
 SUPER .....372  
 SUPER\_JSR .....373  
 SUSPEND\_PR .....374  
 swab .....520  
 swaw .....522  
 swmem .....523  
 symbol .....524  
 SYMBOL .....375  
 system .....526

## T

tan .....527  
 tanh .....527  
 TCOLOR .....376  
 tell .....528  
 tempfile .....529  
 tempnam .....530  
 TEXTGET .....377  
 TEXTPUT .....378  
 TGUSEMD .....379  
 time .....532  
 TIMEASC .....381  
 TIMEBCD .....382  
 TIMEBIN .....384  
 TIMECNV .....385  
 TIMEGET .....386  
 TIME\_PR .....387  
 TIMERDST .....388  
 TIMESET .....389  
 tmasc .....533  
 tmcnv .....534  
 tmpfile .....535  
 tmpnam .....536  
 toascii .....538  
 \_tolower .....538

\_tolower .....538  
 \_toupper .....538  
 toupper .....538  
 TPALET .....390  
 TPALET2 .....391  
 tqsort .....210  
 TRAP15 .....392  
 TVCTRL .....394  
 TXBOX .....396  
 TXFILL .....397  
 TXRASCOPY .....398  
 TXREV .....399  
 TXXLINE .....400  
 TXYLINE .....401  
 tzset .....540

## U

uitoa .....542  
 ultoa .....543  
 ungetc .....544  
 ungetch .....546  
 unlink .....547  
 UNLOCK .....402  
 using .....548  
 utime .....549  
 uwtoa .....550

## V

va\_arg .....551  
 va\_end .....551  
 val .....553  
 va\_start .....551  
 vprintf .....555  
 v\_cut .....554  
 VDISPST .....403



# 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
blbmem	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散する	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
sizemem	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
uwtoa	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
strsfm	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
tmasc	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置を求める	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	255
fputc	ストリームへ 1 文字書き込む	256
fputchar	標準出力へ 1 文字書き込む	245
fputs	ストリームへ文字列を書き込む	267
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する	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	ファイルへデータを書き込む	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	浮動少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
strrchr	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
strmf	ファイル名の拡張子を置き換える	499
strmf	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
strsr	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化する	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_linput	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_sstat	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
CLIPPUP	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行上へスクロールする	86
B_DOWN	n行下へスクロールする	37
B_UP	n行上へスクロールする	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
TIMESSET	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
CRTCAS	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_DEFPCG	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

TXYLINE	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
KEEPFR	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	ブレイ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	ファイルを開く	311
CLOSE	ファイルを閉じる	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か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か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製する	174
DUP2	ファイルハンドルを強制複製する	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	158
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
VERIFYG	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





## **シャープ株式会社**

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

AVCシステム事業推進室

〒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 電子機器事業本部 AVCシステム事業推進室 ソフトウェア担当