C言語プログラミング能力認定試験

１級実技試験

受験者用リファレンス

［第１版］

サーティファイ
情報処理能力認定委員会

目次

１．演算子一覧

優先順位	結合規則	演算子	演算子名	記述形式
1	左から右	() [] . -> ++ --	関数呼出し 配列添字 直接メンバ 間接メンバ 後置増分 後置減分	opr1(opr2) opr1[opr2] opr1.opr2 opr1->opr2 opr1++ opr1--
2	右から左	++ -- sizeof & * + - ~ !	前置増分 前置減分 記憶量 アドレス 間接参照 正符号 負符号 補数 否定	++opr1 --opr1 sizeof opr1 &opr1 *opr1 +opr1 -opr1 ~opr1 !opr1
3	右から左	(type)	キャスト	(type)opr1
4	左から右	* / %	乗算 除算 剰余	opr1 * opr2 opr1 / opr2 opr1 % opr2
5	左から右	+ -	加算 減算	opr1 + opr2 opr1 - opr2
6	左から右	<< >>	左シフト 右シフト	opr1 << opr2 opr1 >> opr2
7	左から右	< <= > >=	左不等 等価左不等 右不等 等価右不等	opr1 < opr2 opr1 <= opr2 opr1 > opr2 opr1 >= opr2
8	左から右	== !=	等価 非等価	opr1 == opr2 opr1 != opr2
9	左から右	&	ビット積	opr1 & opr2
10	左から右	^	ビット差	opr1 ^ opr2
11	左から右	|	ビット和	opr1 | opr2
12	左から右	&&	積結合	opr1 && opr2
13	左から右	||	和結合	opr1 || opr2
14	右から左	?:	条件	opr1 ? opr2 : opr3
15	右から左	= += -= *= /= %= <<= >>= &= ^= |=	単純代入 加算代入 減算代入 乗算代入 除算代入 剰余代入 左シフト代入 右シフト代入 ビット積代入 ビット差代入 ビット和代入	opr1 = opr2 opr1 += opr2 opr1 -= opr2 opr1 *= opr2 opr1 /= opr2 opr1 %= opr2 opr1 <<= opr2 opr1 >>= opr2 opr1 &= opr2 opr1 ^= opr2 opr1 |= opr2
16	左から右	,	順次	opr1 , opr2

●優先順位
　演算の順序を決める規則であり，表中の番号が小さいほど優先順位が高い。

●結合規則
　同じ優先順位の演算子における，入れ子の結合方向であり，評価順序とは異なる。

２．主要標準ライブラリ関数

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

返却値	なし	switch (state) {
		case INITIAL: /* 初期状態 */
		　　do_process();
		　　state = EXECUTING;
		　　break;
		case EXECUTING: /* 実行中 */
		　　busy_process();
		　　break;
		default: /* 状態遷移異常時 */
		　　abort();
		}

説　明	プロセスを意図的に異常終了させる。UNIX 系の OS では，そのときのメモリの内容が（コアダンプファイルという）ファイルとして出力され，プログラム実行時の変数値が参照できるので，デバッグに有効である。	

abs	#include <stdlib.h> int abs(int n);	

返却値	計算結果	#include <stdio.h>
		#include <stdlib.h>
		
		int main(void)
		{
		　　int n;
		　　for (n = -10; n <= 10; n++) {
		　　　　printf("abs(%d) = %d¥n", n, abs(n));
		　　}
		　　return 0;
		}

説　明	n の絶対値を返す。	

acos	#include <math.h> double acos(double x);	

返却値	計算結果	#include <stdio.h>
		#include <math.h>
		
		int main(void)
		{
		　　printf("3 * acos(1.0/2.0) = %f¥n",
		　　　　3 * acos(1.0/2.0));
		　　return 0;
		}
		
		【実行結果】
		3 * acos(1.0/2.0) = 3.141593

説　明	アークコサインを返す。	

asctime	#include <time.h> char *asctime(const struct tm *time);	

返却値	変換した文字列へのポインタ	#include <stdio.h>
		#include <time.h>
		
		int main(void)
		{
		　　time_t now;
		　　struct tm *local_time;
		
		　　time(&now);
		　　local_time = localtime(&now);
		　　printf("%s", asctime(local_time));
		　　return 0;
		}
		
		【実行結果】
		Thu Nov 29 02:18:01 2012

説　明	構造体 tm に格納されている時刻を，特定の形式の文字列に変換する。変換した文字列の最後にはナル文字'¥n'が格納される。 asctime は，time 関数で得られたデータを localtime または gmtime 関数で tm 型構造体に変換したものを特定の形式の文字列に変換する。構造体 tm，time_t 型については“処理系で定義される型”を参照。 time，localtime，gmtime 関数参照。	

asin — `#include <math.h> double asin(double x);`

返却値: 計算結果

説　明: アークサインを返す。

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

int main(void)
{
    printf("6 * asin(1.0/2.0) = %f¥n",
        6 * asin(1.0/2.0));
    return 0;
}
```

【実行結果】

```
6 * asin(1.0/2.0) = 3.141593
```

assert — `#include <assert.h> void assert(int test);`

返却値: なし

説　明: test には条件式を書く。条件が偽のとき，診断メッセージを出力し，abort 関数を呼び出して処理を中断する。診断メッセージは「メッセージ，条件式，ファイル名，行番号」が含まれる。assert はプログラムのデバッグ時に使用する。NDEBUG マクロが定義されていると assert は無効になる。このため完成プログラムに assert 記述を残すことができる。「assert(条件)」は「条件が真なら次へ進め」という意味をもつ。assert はマクロで実現される。次に assert マクロの定義例を示す。

```
#ifdef NDEBUG
#define assert(exp)((void)0)
#else
#define assert(exp)(void)((exp) ||
 (_assert(#exp,__FILE__,__LINE__),0))
#endif /* NDEBUG */
```

ここで，“__FILE__”マクロは，プログラムファイルが格納されているファイル名を，また，“__LINE__”マクロは，行番号を求めるマクロである。

【用例１】asstst.c というプログラムでテストする。

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

int main(int argc, char *argv[])
{
    printf("開始します¥n");
    assert(argc!=1);
    printf("終了します¥n");
    return 0;
}
```

【実行結果】

```
A>asstst
開始します
Assertion failed: argc!=1, file asstst.c, line 7

abnormal program termination
```

【用例２】同じプログラムで NDEBUG 指定をする。

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

int main(int argc, char *argv[])
{
    printf("開始します¥n");
    assert(argc!=1);
    printf("終了します¥n");
    return 0;
}
```

【実行結果】

```
A>asstst
開始します
終了します
```

atan — `#include <math.h> double atan(double x);`

返却値: 計算結果

説　明: アークタンジェントを返す。

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

int main(void)
{
    printf("6 * atan(1.0/1.7320508) = %f¥n",
        6 * atan(1.0/1.7320508));
    return 0;
}
```

【実行結果】

```
6 * atan(1.0/1.7320508) = 3.141593
```

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

返却値: 計算結果

説　明: y/x のアークタンジェントを返す。

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

int main(void)
{
     printf("6 * atan2(1.0, 1.7320508) = %f¥n",
          6 * atan2(1.0, 1.7320508));
     return 0;
}
```

【実行結果】

```
6 * atan2(1.0, 1.7320508) = 3.141593
```

atexit — #include <stdlib.h> int atexit(void (*func)(void));

返却値: 正常時：0　エラー時：非 0

説　明: プログラム終了時に自動実行される関数 func を登録する。関数 func は引数も返却値もない関数でなければならない。

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

void finish(void);
int *a;

int main(void)
{
     int i;
     a = (int *)malloc(1024 * sizeof(int));
     printf("メモリ確保¥n");
     if (atexit(finish) != 0) {
          puts("finish():登録に失敗しました");
          exit(1);
     }
     return 0;
}

void finish(void)
{
     free(a);
     a = NULL;
     printf("メモリ解放¥n");
}
```

【実行結果】

```
メモリ確保
メモリ解放
```

atof — #include <stdlib.h> double atof(const char *str);

返却値: 変換された値
変換できない（数字以外の文字やオーバフローなどの）ときの動作は定義されない（処理系に依存）。

説　明: 文字列 str を double 型変数に変換する。

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

int main(void)
{
     const char *str = "3.141593";
     printf("atof(%s) = %lf¥n",
               str, atof(str));
     return 0;
}
```

【実行結果】

```
atof(3.141593) = 3.141593
```

atoi — `#include <stdlib.h> int atoi(const char *str);`

返却値	変換された値 変換できない（数字以外の文字やオーバフローなどの）ときの動作は定義されない（処理系に依存）。
説　明	文字列 str を int 型変数に変換する。

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

int main(void)
{
     const char *str = "1234";
     printf("atoi(%s) = %d¥n",
             str, atoi(str));
     return 0;
}
```

【実行結果】

```
atoi(1234) = 1234
```

atol — `#include <stdlib.h> long int atol(const char *str);`

返却値	変換された値 変換できない（数字以外の文字やオーバフローなどの）ときの動作は定義されない（処理系に依存）。
説　明	文字列 str を long 型変数に変換する。

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

int main(void)
{
     const char *str = "1234567890";
     printf("atol(%s) = %ld¥n",
             str, atol(str));
     return 0;
}
```

【実行結果】

```
atol(1234567890) = 1234567890
```

bsearch — `#include <stdlib.h>  void *bsearch(const void *key, const void *base, size_t n, size_t size, int (*cmp)(const void *arg1, const void *arg2));`

返却値	正常時：マッチした項目へのポインタ マッチしないとき：NULL
説　明	バイナリサーチ（二分探索）を行う。 base[0]～base[n-1]の中から key で示すデータを探す。一つの要素は size バイト長で示す。配列 base の中身は昇順に整列されていなければならない。 関数 cmp は次の返却値をもつものでなければならない。

第 1 引数 ＞ 第 2 引数	正
第 1 引数 ＝ 第 2 引数	0
第 1 引数 ＜ 第 2 引数	負

```
#include <stdio.h>
#include <stdlib.h>
int intcmp(const void *a, const void *b);

int main(void)
{
     int entry[] = {1,2,4,7,9,1,17,22,24,29,33};
     int N = sizeof(entry) / sizeof(int);
     int n = 17;
     p = (int *)bsearch(&n,
                        entry,
                        N,
                        sezeof(int),
                        intcmp
                      );
     if (p == NULL) printf("登録されていない¥n")
     else printf("%d は登録されている¥n", *p);
     return 0;
}

int intcmp(const void *a, const void *b)
{
     return *(int *)a - *(int *)b;
}
```

【実行結果】

```
17 は登録されている
```

calloc — #include <stdlib.h> void *calloc(size_t n, size_t size);

返却値：正常時：割り当てたメモリへのポインタ
エラー時：NULL

説　明：size バイトのメモリ領域を n 個分確保し，それへのポインタを返す。確保した領域は 0 に初期化される。

```
int *pi;
pi=(int *)calloc(128, sizeof(int));
if(pi==NULL){
    printf("メモリ確保失敗");
}
```

ceil — #include <math.h> double ceil(double x);

返却値：計算結果

説　明：小数点以下の数値を切り上げる。

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

int main(void)
{
    printf("ceil(%f) = %f¥n",
        12.34, ceil(12.34));
    printf("ceil(%f) = %f¥n",
        -56.78, ceil(-56.78));
    return 0;
}
```

【実行結果】

```
ceil(12.340000) = 13.000000
ceil(-56.780000) = -56.000000
```

clearerr — #include <stdio.h> void clearerr(FILE *fp);

返却値：なし

説　明：ファイル fp のエラーフラグと EOF フラグをクリアする。

```
if(ferror(fp)){
    printf("Error.¥n");
    (エラー時の処理)
    clearerr(fp);
}
```

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

返却値：正常値：経過時間
エラー時：(clock_t)(-1)

説　明：プログラム実行開始からの経過時間を得る。clock 関数を 2 回呼び出し，その返却値の差を求める。これを CLOCKS_PER_SEC で割ると秒単位にすることができる。clock_t 型については“処理系で定義される型”を参照。

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

int main(void)
{
    clock_t tm1, tm2;
    int i;

    for (i = 1; i <= 20; i++){
        tm1 = clock();
        while(!0){
            tm2 = clock();
            if ((tm2 - tm1)/CLOCKS_PER_SEC >= 1)
                break;
        }
        printf("¥abeep!!¥n");
    }
    return 0;
}
```

cos — #include <math.h> double cos(double x);

返却値：計算結果

説　明：コサイン値を返す。

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

int main(void)
{
    printf("cos(pi/3) = %f¥n", cos(3.141593/3));
    return 0;
}
```

【実行結果】

```
cos(pi/3) = 0.500000
```

cosh	#include <math.h> double cosh(double x);	

返却値	計算結果	#include <stdio.h>
		#include <math.h>
		
		int main(void)
		{
		printf("cosh(0) = %f¥n", cosh(0));
		return 0;
		}
		
		【実行結果】
		cosh(0) = 1.000000

説　明	ハイパボリックコサイン値を返す。	
ctime	#include <time.h> char *ctime(const time_t *timeptr);	

返却値	変換した文字列へのポインタ	#include <stdio.h>
		#include <time.h>
		
		int main(void)
		{
		time_t now;
		
		time(&now);
		printf("%s", ctime(&now));
		return 0;
		}
		
		【実行結果】
		Sun Jun 07 18:44:04 1998

説　明	time 関数で取得した暦時刻 timeptr を，現地時間（日本時間）の特定の形式の文字列に変換する。変換した文字列の最後のナル文字'¥0'の一つ前に'¥n'が格納される。time_t 型については“処理系で定義される型”を参照。 ctime 関数は「localtime + asctime」と同じである。	
difftime	#include <time.h> difftime(time_t time1, time_t time2);	

返却値	得られた時間差	時間間隔を測定する。
		#include <stdio.h>
		#include <time.h>
		
		int main(void)
		{
		time_t time1, time2;
		char ss[80];
		
		time(&time1)
		printf("測定開始:[Enter]を押下してください");
		gets(ss);
		time(&time2);
		printf("測定終了:%f 秒経過しました.¥n",
		difftime(time2, time1));
		return 0;
		}
		
		【実行結果】
		測定開始:[Enter]を押下してください
		測定終了:6.000000 秒経過しました

説　明	time1 - time2 の結果を秒単位で計算する。time1 - time2 は time 関数で取得した暦時刻である。time_t 型については“処理系で定義される型”を参照。	
div	#include <stdlib.h> div_t div(int a, int b);	
返却値	計算結果を入れた構造体	div_t div1; div1=div(10, 46); printf("商%d¥n 余り=%d¥n" div1.quot, div1.rem);
説　明	a÷b を計算し，その商（=quotient）を div_t 型のメンバ quot に，余り（=remainder）を div_t 型のメンバ rem に入れる。構造体 div_t については“処理系で定義される型”を参照。	

exit	#include <stdlib.h> void exit(int status);	

返却値	なし	exit(EXIT_FAILURE);
		または
		exit(1);

説　明	オープンされているすべてのファイルを正常にクローズして，プログラムを終了する。そのとき，数値 status を終了コードとして親プロセスに返す。終了コードは，次の標準値が stdlib.h の中で定義されている。 EXIT_SUCCESS → 正常終了 EXIT_FAILURE → 異常終了 実際には，EXIT_SUCCESS=0，EXIT_FAILURE=1 と定義されている。このため通常は正常終了に exit(0)を，異常終了に exit(1)を用いることが多い。	
exp	#include <math.h> double exp(double x);	
返却値	計算結果	double da; da = exp(2.5);
説　明	指数関数 e の x 乗を返す。	
fabs	#include <math.h> double fabs(double x);	
返却値	計算結果	double da; da = fabs(-30.1);
説　明	double 型データ x の絶対値を返す。	
fclose	#include <stdio.h> int fclose(FILE *fp);	
返却値	正常時：0　エラー時：EOF	FILE *fp; (ファイル処理) fclose(fp);
説　明	ファイル fp をクローズする。	
feof	#include <stdio.h> int feof(FILE *fp);	

返却値	ファイル終了：非 0　終了でない：0	while (!feof(fp)) {
		(fp を使ったファイルの処理)
		…
		}

説　明	ファイル fp の EOF フラグをチェックする。	
ferror	#include <stdio.h> int ferror(FILE *fp);	

返却値	エラーが発生している：非 0　エラーはない：0	if (ferror(fp)) {
		(エラー時の処理)
		clearerr(fp);
		}

説　明	ファイル fp の読み書きエラーをチェックする。	
fflush	#include <stdio.h> int fflush(FILE *fp);	
返却値	正常時：0　エラー時：EOF	fflush(fp);
説　明	現在バッファに入っているすべての文字をファイル fp に書き出す。	
fgetc	#include <stdio.h> int fgetc(FILE *fp);	
返却値	正常時：読み込んだ文字　ファイル終了時：EOF	int ca; ca = fgetc(fp);
説　明	ファイル fp から 1 文字読み込む。	
fgetpos	#include <stdio.h> int fgetpos(FILE *fp, fpos_t *pos);	
返却値	正常時：0　エラー時：非 0	fsetpos 参照。
説　明	ファイル fp のファイル位置指示子の値を pos の示す場所に入れる。その値は fsetpos 関数で用いる。fgetpos～fsetpos を用いることで，同じ位置から読み書きを再開できる。fpos_t 型については“処理系で定義される型”を参照。	

<table>
<tr><td>fgets</td><td colspan="2">#include <stdio.h> char *fgets(char *buf, int maxchar, FILE *fp);</td></tr>
<tr><td>返却値</td><td>正常時：ポインタ buf そのもの
ファイル終了／エラー時：NULL</td><td rowspan="3">/* fgets.c */
#include <stdio.h>

int main(void)
{
　　FILE *stream = fopen("fgets.c", "r");
　　char buf[1024];
　　while (fgets(buf, sizeof(buf), steam)!=NULL)
　　　　fputs(buf, stdout);
　　fclose(stream);
　　return 0;
}

【実行結果】
/* fgets.c */
#include <stdio.h>

int main(void)
{
　　FILE *stream = fopen("fgets.c", "r");
《以下省略》</td></tr>
<tr><td>説　明</td><td>ファイル fp から maxchar-1 文字まで，または復改文字までの文字列を読み込む。読み込んだ文字列の最後にナル文字'¥0'を付加する。復改文字もそのまま読み込む。</td></tr>
<tr><td>floor</td><td colspan="2">#include <math.h> double floor(double x);</td></tr>
<tr><td>返却値</td><td>計算結果</td><td rowspan="2">double da;
da = floor(-7.19);</td></tr>
<tr><td>説　明</td><td>数値を切り下げる。</td></tr>
<tr><td>fmod</td><td colspan="2">#include <math.h> double fmod(double x, double y);</td></tr>
<tr><td>返却値</td><td>計算結果</td><td rowspan="2">double da;
da = fmod(51.2, 1.8);</td></tr>
<tr><td>説　明</td><td>x/y の余りを計算する。</td></tr>
<tr><td>fopen</td><td colspan="2">#include <stdio.h> FILE *fopen(const char *filename, const char *openmode);</td></tr>
<tr><td>返却値</td><td>正常時：オープンしたファイルへのポインタ
エラー時：0</td><td rowspan="2">fgets()を参照。</td></tr>
<tr><td>説　明</td><td>filename で示すファイルを，openmode で示すモードでオープンする。オープンモードの例。
<table><tr><td>"r"</td><td>読込み</td></tr><tr><td>"w"</td><td>書出し</td></tr><tr><td>"a"</td><td>追加書出し</td></tr></table></td></tr>
<tr><td>fprintf</td><td colspan="2">#include <stdio.h> int fprintf(FILE *fp, const char *format, ...);</td></tr>
<tr><td>返却値</td><td>正常時：出力した文字数　エラー時：負の値</td><td rowspan="2">fprintf(fp, "%d + %d¥n", 30, 21);</td></tr>
<tr><td>説　明</td><td>対応する引数の内容を，format で指定する書式に従ってファイル fp に出力する。</td></tr>
<tr><td>fputc</td><td colspan="2">#include <stdio.h> int fputc(int c, FILE *fp);</td></tr>
<tr><td>返却値</td><td>正常時：出力した文字　エラー時：EOF</td><td rowspan="2">fputc('x', fp);</td></tr>
<tr><td>説　明</td><td>文字 c をファイル fp に書き出す。</td></tr>
<tr><td>fputs</td><td colspan="2">#include <stdio.h> int fputs(char *str, FILE *fp)</td></tr>
<tr><td>返却値</td><td>正常時：非負　エラー時：EOF</td><td rowspan="2">char *str = "string";
fputs(str, fp);</td></tr>
<tr><td>説　明</td><td>文字列 str をファイル fp に書き出す。復改文字を付加しない。</td></tr>
<tr><td>fread</td><td colspan="2">#include <stdio.h> size_t fread(void *buf, size_t size, size_t n, FILE *fp);</td></tr>
<tr><td>返却値</td><td>読み込んだデータの個数。返却値が n より小さいときはファイル終了かエラー発生である。</td><td rowspan="2">FILE *fp;
char buf[128];
（fopen などのファイル入出力前処理）
fread(buf, 1, 3, fp);</td></tr>
<tr><td>説　明</td><td>ファイル fp から「size バイト×n 個」分のデータを buf に読み込む。size_t は stdio.h の中で定義されている。</td></tr>
<tr><td>free</td><td colspan="2">#include <stdlib.h> void free(void *ptr);</td></tr>
<tr><td>返却値</td><td>なし</td><td rowspan="2">atexit()を参照。</td></tr>
<tr><td>説　明</td><td>malloc，calloc，realloc によって確保された ptr で示されるメモリ領域を解放する。</td></tr>
</table>

<table>
<tr><th>freopen</th><td colspan="2">#include <stdio.h> FILE *freopen(const char *filename, const char *openmode, FILE *fp);</td></tr>
<tr><td>返却値</td><td>正常時：オープンしたファイルへのポインタ
エラー時：NULL</td><td rowspan="2">stdout を tst.dat ファイルに割り付ける。"string1"は画面に，"string2"は tst.dat に書き出される。
#include <stdio.h>
int main(void)
{
printf("string1¥n");
if (freopen("tst.dat", "w",stdout) == NULL){
fprintf(stderr, "リオープンできません¥n");
exit(1);
}
printf("string2¥n");
return 0;
}</td></tr>
<tr><td>説　明</td><td>現在 fp で示されるオープン済みのファイルをクローズする。新たに filename で示すファイルを openmode で指定のモードでオープンして，fp に再割り付けする。オープンモードは fopen と同じ。この関数を用いると標準入出力である stdin，stdout，stderr を他のファイルに割り付けることができる。</td></tr>
<tr><th>frexp</th><td colspan="2">#include <math.h> double frexp(double value, int *p);</td></tr>
<tr><td>返却値</td><td>計算結果</td><td rowspan="2">double da;
int pi;
da = frexp(4.0, &pi);</td></tr>
<tr><td>説　明</td><td>実数（浮動小数点数）を小数 x と指数 n に分解する。それぞれの値は，value = x*2^n となる仮数部 x と指数 n である。仮数部 x を返却値とし，指数部 n は p で示すアドレスに入る。x は 0.5 以上，1.0 未満の値をとる。</td></tr>
<tr><th>fscanf</th><td colspan="2">#include <stdio.h> int fscanf(FILE *fp, const char *format, ...);</td></tr>
<tr><td>返却値</td><td>正常時：入力データの個数
ファイル終了／エラー時：EOF</td><td rowspan="2">FILE fp;
int ia;
…　　(fopen など，前処理)
fscanf(fp, "%d", &ia);</td></tr>
<tr><td>説　明</td><td>ファイル fp から，書式 format に従って対応する引数にデータを読み込む。</td></tr>
<tr><th>fseek</th><td colspan="2">#include <stdio.h> int fseek(FILE *fp, long offset, int origin);</td></tr>
<tr><td>返却値</td><td>正常時：0　エラー時：非 0</td><td rowspan="2">FILE fp;

fseek(fp, 20L, SEEK_SET)
fseek(fp, 4L, SEEK_CUR);
fseek(fp, -3L, SEEK_END)</td></tr>
<tr><td>説　明</td><td>ファイル fp ファイルポジションを，origin を基点として offset バイト移動する。offset をマイナス値にすればファイルの先頭方向に移動する。origin(移動の基点)の指定方法は，以下のとおりである。
<table><tr><td>SEEK_SET</td><td>先頭位置から</td></tr><tr><td>SEEK_CUR</td><td>現在位置から</td></tr><tr><td>SEEK_END</td><td>終端位置から</td></tr></table></td></tr>
<tr><th>fsetpos</th><td colspan="2">#include <stdio.h> int fsetpos(FILE *fp, const fpos_t *pos);</td></tr>
<tr><td>返却値</td><td>正常時：0　エラー時：非 0</td><td rowspan="2">fpos_t ps;
…
fgetpos(fp, &ps);
（データ読込み）
fsetpos(fp, &ps)
（同じデータを再読込み）</td></tr>
<tr><td>説　明</td><td>ファイル fp のファイル位置指示子を pos でセットし直す。pos の値は，あらかじめ fgetpos 関数で設定しておいたものを用いる。fgetpos～fsetpos を用いることで，同じ位置から読み書きを再開できる。fpos_t 型については“処理系で定義される型”を参照。</td></tr>
<tr><th>ftell</th><td colspan="2">#include <stdio.h> long ftell(FILE *fp);</td></tr>
<tr><td>返却値</td><td>正常時：現在のファイルポジション
エラー時：-1L</td><td rowspan="2">long lps;
lps = ftell(fp);
（ファイル読書き）
fseek(fp, lps, SEEK_SET);</td></tr>
<tr><td>説　明</td><td>ファイル fp の現在のファイルポジションを返す。この値は fseek 関数で offset(移動量)として使うことができる。</td></tr>
<tr><th>fwrite</th><td colspan="2">#include <stdio.h> size_t fwrite(const void *buf, size_t size, size_t n, FILE *fp);</td></tr>
<tr><td>返却値</td><td>正しく書き出したデータの個数。返却値が N より小さいときはエラー発生である。</td><td rowspan="2">#define SIZE 16
…
fwrite(buf, SIZE, N,fp);</td></tr>
<tr><td>説　明</td><td>buf の先頭から「SIZE バイト×N 個」分のデータをファイル fp に書き出す。</td></tr>
<tr><th>getc</th><td colspan="2">#include <stdio.h> int getc(FILE *fp);</td></tr>
<tr><td>返却値</td><td>正常時：読み込んだ文字
ファイル終了時：EOF</td><td rowspan="2">int ca;
ca = getc(fp);</td></tr>
<tr><td>説　明</td><td>ファイル fp から 1 文字読み込む。マクロで実現されていることがあること以外は fgetc と同じ。</td></tr>
</table>

getchar	#include <stdio.h> int getchar(void);	
返却値	正常時：読み込んだ文字 ファイル終了／エラー時：EOF	int ca; ca = getchar();
説　明	stdin から 1 文字を読み込む。	
getenv	#include <stdlib.h> char *getenv(const char *env);	

返却値	正常時：該当する文字列へのポインタ
	環境変数が見つからないとき：NULL
		char *path;
		path = getenv("PATH");
		if (path != NULL)
		　printf("%s¥n", path);

説　明	env で示す環境変数の値を得る。	
gets	#include <stdio.h> char *gets(char *str);	
返却値	正常時：引数 str ファイル終了/エラー時：NULL	char str[128]; gets(str);
説　明	stdin から文字列を入力し str に格納する。復改文字は捨てる。バッファオーバーランエラーによるセキュリティーホールの原因となるので，fgets()を使用したほうがよい。	
gmtime	#include <time.h> struct tm *gmtime(const time_t *timeptr);	
返却値	変換した時刻のポインタ　エラー時：NULL	日本時間と世界標準時間を比較する。(地域時間差を設定済みとする。)

説　明	time_t 型の暦時刻を構造体 tm 型のグリニッジ標準時（世界標準時）に変換する。localtime 関数は現地時間（日本時間）を得る点が異なる。日本とは 9 時間ずれている(日本が進んでいる)。地域時間差の設定は処理系に依存する。構造体 tm，time_t 型については localtime 関数を参照。	#include <stdio.h>
		#include <time.h>
		
		int main(void)
		{
		　time_t now;
		　struct tm *lt;
		　struct tm *gt;
		
		　time(&now);
		　lt = localtime(&now);
		　printf("日本時間:%s", asctime(lt));
		　gt = gmtime(&now);
		　printf("標準時間:%s", asctime(gt));
		　return 0;
		}
		
		【実行結果】
		日本時間:Sun Jun 07 17:56:01 1998
		標準時間:Sun Jun 07 08:56:01 1998

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

返却値	英数字のとき：非 0　そうでないとき：0	for (c = 0; c <= 255; c++)
		　if (isalnum(c))
		　　printf("¥'%c¥'は英数字です。¥n", c);

説　明	文字 c が英数字（A～Z，a～z，0～9）なら真を返す。	
isalpha	#include <ctype.h> int isalpha(int c);	

返却値	英字のとき：非 0　そうでないとき：0	for (c = 0; c <= 255; c++)
		　if (isalpha(c))
		　　printf("¥'%c¥'は英字です。¥n", c);

説　明	文字 c が英文字（A～Z，a～z）なら真を返す。	
iscntrl	#include <ctype.h> int iscntrl(int c);	

返却値	制御文字のとき：非 0　そうでないとき：0	for (c = 0; c <= 255; c++)
		　if (iscntrl(c))
		　　printf("¥'0x%02¥'は制御文字です。¥n", c);

説　明	文字 c が制御文字（0x00～0x1F，0x7F）なら真を返す。	
isdigit	#include <ctype.h> int isdigit(int c);	

返却値	数字のとき：非 0　そうでないとき：0	for (c = 0; c <= 255; c++)
		　if (isdigit(c))
		　　printf("¥'%c¥'は数字です。¥n", c);

説　明	文字 c が数字（0～9）なら真を返す。	
isgraph	#include <ctype.h> int isgraph(int c);	

返却値	空白以外の印字可能文字のとき：非 0
	そうでないとき：0
		for (c = 0; c <= 255; c++)
		　if (isgraph(c))
		　　printf("¥'%c¥'は空白以外の印字可能文字です。¥n", c);

説　明	文字 c が空白を除く印字可能文字（0x21～0x7E）なら真を返す。	

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

返却値	小文字のとき：非 0　そうでないとき：0	for (c = 0; c <= 255; c++)
		　if (islower(c))
		　　printf("¥'%c¥'は小文字です。¥n", c);

説　明	文字 c が小文字（a～z）なら真を返す。	
isprint	#include <ctype.h> int isprint(int c);	

返却値	印字可能文字のとき：非 0
	そうでないとき：0
		for (c = 0; c <= 255; c++)
		　if (isprint(c))
		　　printf("¥'%c¥'は印字可能文字です。¥n", c);

説　明	文字 c が印字可能文字（0x20～0x7E）なら真を返す。	
ispunct	#include <ctype.h> int ispunct(int c);	

返却値	区切り文字のとき：非 0　そうでないとき：0	for (c = 0; c <= 255; c++)
		　if (ispunct(c))
		　　printf("¥'%c¥'は区切り文字です。¥n", c);

説　明	文字 c が区切り文字（punctuation character）なら真を返す。区切り文字とは空白，英数字以外の印字可能文字。	
isspace	#include <ctype.h> int isspace(int c);	

返却値	空白系文字のとき：非 0　そうでないとき：0	for (c = 0; c <= 255; c++)
		　if (isspace(c))
		　　printf("¥'0x%02x¥'は空白系の文字です。¥n", c);

説　明	文字 c が空白，タブ，復帰，改行，垂直タブ，フォームフィード（0x09～0x0D，0x20）なら真を返す。	
isupper	#include <ctype.h> int isupper(int c);	

返却値	大文字のとき：非 0　そうでないとき：0	for (c = 0; c <= 255; c++)
		　if (isupper(c))
		　　printf("¥'%c¥'は大文字です。¥n", c);

説　明	文字 c が大文字（A～Z）なら真を返す。	
isxdigit	#include <ctype.h> int isxdigit(int c);	

返却値	16 進文字のとき：非 0　そうでないとき：0	for (c = 0; c <= 255; c++)
		　if (isxdigit(c))
		　　printf("¥'%c¥'は 16 進文字です。¥n", c);

説　明	文字 c が 16 進文字（0～9，A～F，a～f）なら真を返す。	
labs	#include <stdlib.h> long labs(long n);	
返却値	n の絶対値	long l_val = -123456789L; printf("%ld¥n", labs(l_val));
説　明	long 型データ n の絶対値を返す。	
ldexp	#include <math.h> double ldexp(double x, int n);	
返却値	計算結果	double da; da = ldexp(1.2, 2);
説　明	x * 2^n となる値を返す。	
ldiv	#include <stdlib.h> ldiv_t ldiv(long a, long b);	
返却値	計算結果を入れた構造体を返す。	ldiv_t ldat; ldat = ldiv(1234567890L, 56873L); printf("商=%ld 余り=%ld¥n", ldat. quot, ldat. rem);
説　明	long 型の値で a÷b を計算し，その商（=quotient）を ldiv_t 型のメンバ quot に，余り（remainder）を ldiv_t 型のメンバ rem に入れる。構造体 ldiv_t 型については“処理系で定義される型”を参照。	

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

返却値	lconv 型構造体へのポインタ	#include <stdio.h>
		#include <locale.h>
		
		int main(int argc, char *argv[])
		{
		struct lconv *lc;
		setlocale(LC_ALL, argv[1]);
		lc=localeconv();
		printf("小数点文字 = [%s]¥n",
		lc->decimal_point);
		printf("通貨記号 = [%s]¥n",
		lc->currency_symbol);
		printf("桁グループ区切子= [%s]¥n",
		lc->thousands_sep);
		printf("負の金額値の符号文字= [%s]¥n",
		lc->negative_sign);
		printf("金額の小数点以下桁数= %d¥n",
		lc->frac_digits);
		return 0;
		}
		
		【実行結果】
		C:¥> local_test jpn
		小数点文字 = [.]
		通貨記号 = [¥]
		桁グループ区切子 = [,]
		負の金額値の符号文字 = [-]
		金額の小数点以下桁数 = 0
		
		C:¥> local_test usa
		小数点文字 = [.]
		通貨記号 = [$]
		桁グループ区切子 = [,]
		負の金額値の符号文字 = [-]
		金額の小数点以下桁数 = 2

説　明	地域で使われる数字や貨幣通貨記号などの表現方法に関する情報を構造体 lconv に初期設定する。	

localtime	#inclode <time.h> struct tm *localtime(const time_t *timeptr);	

返却値	構造体 tm 型へのポインタ	#include <stdio.h>
		#include <time.h>
		
		int main(void)
		{
		time_t now;
		struct tm *local_time;
		
		time(&now);
		local_time=localtime(&now);
		printf("月=%d¥n", local_time->tm_mon + 1);
		printf("日=%d¥n", local_time->tm_mday);
		printf("時=%d¥n", local_time->tm_hour);
		printf("分=%d¥n", local_time->tm_min);
		printf("秒=%d¥n", local_time->tm_sec);
		printf("%s", asctime(local_time));
		return 0;
		}
		
		【実行結果】
		月 = 12
		日 = 1
		時 = 22
		分 = 32
		秒 = 56
		Sun Dec 01 22:32:56 2002

説　明	time 関数で取得した暦時刻 timeptr を構造体型の現地時間（日本時間）に変更する。構造体 tm，time_t 型については“処理系で定義される型”を参照。	

log	#include <math.h> double log(double x);	
返却値	計算結果	double da; da = log(3.1234);
説　明	自然対数（e を底とする対数）を返す。	
log10	#include <math.h> double log10(doble x);	
返却値	計算結果	double da; da = log10(120.0);
説　明	常用対数（10 を底とする対数）を返す。	
longjmp	#include <setjmp.h> void longjmp(jmp_buf env, int val);	
返却値	なし	setjmp 参照。
説　明	setjmp でセーブされた環境情報である env を使って，その setjmp が記述されている位置に大域ジャンプする。そのときに状態値 val を setjmp 関数に渡す。この値が setjmp 関数に渡され，setjmp 関数の返却値となる。その値は 0 以外でなければならない。もし 0 を渡すと setjmp の返却値は 1 に調整される。jmp_buf 型は setjmp.h の中で定義されている。	
malloc	#include <stdlib.h> void *malloc(size_t size);	

返却値	正常時：割り当てたメモリへのポインタ
	エラー時：NULL
		char *cp;
		cp = (char *)malloc(1000);
		if (cp == NULL) {
		　　fprintf(stderr, "メモリ不足です！！¥n");
		　　exit(1);
		}

説　明	size バイトのメモリ領域を確保し，それへのポインタを返す。確保した領域が初期化されることはない。	
mblen	#include <stdlib.h> int mblen(const char *mbstr, size_t n);	

返却値	文字のバイト長。先頭から n 個の文字が正当なマルチバイト文字でないと-1 を返す。	#include <stdlib.h>
		#include <stdio.h>
		
		int main(void)
		{
		　　int　i;
		　　char ambc[MB_CUR_MAX + 1];
		　　char *pmbc = ambc;
		　　wchar_t　wc　= L'a';
		
		　　printf("ワイド文字をマルチバイト文字に変換:¥n");
		　　i = wctomb(pmbc, wc);
		　　printf("¥t 変換する文字のバイト数: %u¥n", i);
		　　printf("¥t マルチバイト文字: %s¥n¥n", pmbc);
		
		　　i = mblen(pmbc, MB_CUR_MAX);
		　　printf("マルチバイト文字%s のバイト数%u¥n", pmbc, i);
		
		　　pmbc = NULL;
		　　i = mblen(pmbc, MB_CUR_MAX);
		　　printf("NULL マルチバイト文字のバイト数%u¥n", i);
		　　return 0;
		}

説　明	マルチバイト文字 mbstr で示す 1 文字の長さを取得し，また正当性を調べる。	
mbstowcs	#include <stdlib.h> size_t mbstowcs (wchar_t *wcstr, const char *mbstr, size_t count);	

返却値	変換したマルチバイト文字数を返す。無効なマルチバイト文字を検出すると-1 を返す。	#inlcude <stdio.h>
		#include <stdlib.h>
		int main(void)
		{
		　　int i;
		　　wchar_t wcstr[] = L"ワイド";
		　　char mbstr[128];
		
		　　i = mbstowcs(wcstr, mbstr, 80);
		
		　　printf("文字列 %s, 文字数 %d¥n", mbstr, i);
		　　return 0;
		}

説　明	マルチバイト文字列を対応するワイド文字列に変換する。	

mbtowc `#include <stdlib.h> int mbtowc (wchar_t *wchar, const char *mbchar, size_t n);`

返却値

マルチバイト文字の長さをバイト単位で返す。正当なマルチバイト文字でないときは-1を返す。

説　明

mbchar で示す最大 n 個のマルチバイト文字を対応するワイド文字に変換する。

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

int main(void)
{
     int i;
     wchar_t wcc;
     char mbs[128];
     wctomb(mbs, L'あ')
     i = mbtowc(&wcc, mbs, 2);
     printf("文字コード%04x,バイト数%d¥n", wcc, i);
     return 0;
}
```

memchr `#include <string.h> void *memchr(const void *buf, int ch, size_t n);`

返却値

見つかったとき：その文字へのポインタ
見つからないとき：NULL

説　明

buf の先頭から n 文字をサーチし，引数 ch のある位置を返す。ナル文字があっても無視してとにかく n 文字サーチする。memchr，memcmp，memcpy，memmove，memset の各関数は指定されたメモリ領域を処理対象にする。処理対象を「文字列」として認識することはないので，ナル文字'¥0'も単なるデータとして扱うだけである。

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

int main(void)
{
     const char *str = "abc¥0def¥0ghi";
     const char *result = memchr(str, 'h', 12);
     puts(result);
     return 0;
}
```

【実行結果】

```
hi
```

memcmp `#include <string.h> int memcmp(const void *buf1, const void *buf2, size_t n);`

返却値

比較結果により以下の値を返す。これは strcmp の返却値と同じである。

buf1 > buf2	正
buf1 = buf2	0
buf1 < buf2	負

説　明

buf1 と buf2 のそれぞれ n バイトのデータを辞書順に比較する。

```
if(memcmp(dt1, dt2, sizeof(int)*10)==0)
     printf("等しい¥n")
```

memcpy `#include <string.h> void *memcpy(void *buf1, const void *buf2, size_t n);`

返却値

コピー後の buf1 を返す

説　明

buf2 の最初の n バイトを buf1 にコピーする。コピー元とコピー先が重なっている場合の動作は不定である。このときは memmove を使うと正しくコピーされる。memcpy は文字列コピーではないので，整数配列の一括コピーにも利用できる。

```
int i;
int ary1[10]={10, 11, 12, 13, 14, 15, 16, 17, 18, 19};
int ary2[10];

memcpy(ary2, ary1, sizeof(ary1));
for(i = 0; i <= 9; i++)
     printf("%d ", ary2[i]);
putchar('¥n');
```

memmove `#include <string.h> void *memmove(void *buf1, const void *buf2, size_t n);`

返却値

コピー後の buf1 を返す

説　明

memcpy と同じ処理をする（buf2 の最初の n バイトを buf1 にコピーする）。memcpy と異なるのは，コピー元とコピー先が重なっている場合にも正しくコピーされることである（その分，遅くなる）。

```
int i;
int ary1[10]={10,11,12,13,14,15,16,17,18,19};

memmove(ary1, ary1 + 2, 8 * sizeof(ary1[0]));
for(i = 0; i <= 9; i++)
     printf("%d ", ary1[i]);
putchar('¥n');
```

memset	#include <string.h> void *memset(void *buf, int ch, size_t n);	

返却値	buf を返す。	#include <stdio.h>
		#include <string.h>
		
		struct color {
		  unsigned char r;
		  unsigned char g;
		  unsigned char b;
		};
		int main(void)
		{
		  struct color black;
		  memset(&black, 0, sizeof(black));
		  printf("(r, g, b) = (%u, %u, %u)¥n",
		  black.r, black.g, black.b);
		  return 0;
		}
		
		【実行結果】
		(r, g, b) = (0, 0, 0)

説　明	buf の指す領域の最初の n バイトを文字 ch で満たす。変数の配列や構造体変数を初期化するのに有効である。	
mktime	#include <time.h> time_t mktime(struct tm *timeptr);	

返却値	正常時：変換された暦時刻のポインタ
	エラー時：(time_t)(-1)
		現在の時間から 3600 秒後の時間を得る
		#include <stdio.h>
		#include <time.h>
		
		int main(void)
		{
		  time_t now;
		  struct tm *l_time;
		
		  time(&now);
		  l_time = localtime(&now);
		  printf("%s¥n", asctime(l_time));
		
		  l_time->tm_sec += 3600;
		
		  now = mktime(l_time);
		  l_time = localtime(&now);
		  printf("%s¥n", asctime(l_time));
		  return 0;
		}

説　明	timeptr の指す構造体 tm 型データを暦時刻に変換する。実行に先立ち構造体 tm の各メンバを設定しておくこと。メンバの値は適正な範囲をオーバーしていても合理的に処理される。たとえば分指定を 65 としたら，時間指定を 1 増やす処理をする。ただし tm_wday と tm_yday は無視され，その値は自動的に計算される。構造体 tm，time_t 型については“処理系で定義される型”を参照。	
modf	#include <math.h> double modf(double x, double *ip);	
返却値	計算結果	double da, db; db = modf(20.1, &da);
説　明	浮動小数点値 x の整数部を ip の指すアドレスに入れ，小数部を返す。	

<table>
<tr><th>perror</th><td colspan="2">#include <stdio.h> void perror(const char *str);</td></tr>
<tr><td>返却値</td><td>なし</td><td rowspan="3">/* mytype.c */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
　　FILE *straem;
　　int c;
　　if ((argc != 2)) {
　　　fprintf(stderr,
　　　　"使用法：%s input.txt",
　　　　argv[0]);
　　　exit(1);
　　}
　　stream = fopen(argv[1], "r");
　　if (stream == NULL) {
　　　perror(argv[1]);
　　　exit(1);
　　}
　　while ((c = getc(stream)) != EOF) {
　　　putchar(c);
　　}
　　fclose(stream);
　　return 0;
}

【実行結果】
C:¥>mytype zettai.nai
zettai.nai:No such file or directory

C:¥>mytype mytype.c
/* mytype.c */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
《以下省略》</td></tr>
<tr><td>説　明</td><td>stderr にエラーメッセージ str を出力する。str をまず表示し，次にコロンに続いて errno に対応するシステムエラーメッセージを表示する。errno はシステムがもっている大域変数で，その値はエラー発生時に一部の標準関数で設定される。この errno を自分で参照するときは errno.h を取り込む。ユーザー自身の表示したいメッセージがないときは str の代わりに perror(NULL); と書く。</td></tr>
<tr><td></td><td></td></tr>
<tr><th>pow</th><td colspan="2">#include <math.h> double pow(double x, double y);</td></tr>
<tr><td>返却値</td><td>計算結果</td><td rowspan="2">double da;
da = pow(1.2, 1.5);</td></tr>
<tr><td>説　明</td><td>x の y 乗を計算する。</td></tr>
<tr><th>printf</th><td colspan="2">#include <stdio.h> int printf(const char *format,...);</td></tr>
<tr><td>返却値</td><td>正常時：出力した文字数　エラー時：負の値</td><td rowspan="2">int id;
double fd;
char sd[80];
　　…
printf("abcde¥n");
printf("%d¥n", id);
printf("%8d¥n", id);
printf("%f¥n", fd);
printf("%s¥n", sd);</td></tr>
<tr><td>説　明</td><td>対応する引数の内容を，format で指定する書式に従って stdout に出力する。</td></tr>
<tr><th>putc</th><td colspan="2">#include <stdio.h> int putc(int c, FILE *fp);</td></tr>
<tr><td>返却値</td><td>正常時：出力した文字　エラー時：EOF</td><td rowspan="2">FILE *fp;
　　…
putc('x', fp);</td></tr>
<tr><td>説　明</td><td>文字 c をファイル fp に出力する。マクロとして実現されることがあること以外は fputc と同じ。</td></tr>
<tr><th>putchar</th><td colspan="2">#include <stdio.h> int putchar(int c);</td></tr>
<tr><td>返却値</td><td>正常時：引数 c　エラー時：EOF</td><td rowspan="2">perror()を参照。</td></tr>
<tr><td>説　明</td><td>文字 c を stdout に出力する。</td></tr>
</table>

puts	#include <stdio.h> int puts(char *str);	
返却値	正常時：非負　エラー時：EOF	memchr()を参照。
説　明	文字列 str を stdout へ出力する。文字列を出力した後に復改を行う。	

qsort	#include <stdlib.h> void qsort(void *base, size_t n, size_t size, int (*cmp)(const void *arg1, const void *arg2));	
返却値	なし	

説　明	クイックソートを行う。base は配列の先頭アドレス，n はデータの要素数，size は一つの要素のバイト数，cmp は比較用関数。qsort 関数は，base[0]～base[n-1]の，1 要素 size バイトのデータをソートする。そのとき要素の大小比較用にユーザー提供関数 cmp を使用する。要素の大小比較用関数 cmp については bsearch の項を参照。	#include <stdio.h>
		#include <stdlib.h>
		int mycmp(const int *, const int *);
		
		int main(void)
		{
		int i;
		int nums[5] =
		{21, 30, 50, 22, 42};
		qsort(nums, 5, sizeof(int), mycmp);
		for (i = 0; i < 5; i++)
		printf("%d", nums[i]);
		printf("¥n");
		return 0;
		}
		int mycmp(const int *x, const int *y)
		{
		return (*x - *y);
		}

raise	#include <signal.h> int raise(int sig);	

返却値	正常時：0　エラー時：非 0	#include <stdio.h>
		#include <signal.h>
		
		int main(void)
		{
		
		raise(SIGABRT);
		
		return 0;
		}

説　明	実行しているプログラムに sig で指定したシグナルを送る。sig には通常 signal.h で定義されている記号定数を指定する。実行中のプログラムは sig に対応した標準の動作（シグナル処理）をする。sig 対応の動作が signal 関数で指定されているときは，その処理を行う。記号定数には少なくとも以下のものがある。これらはすべて signal.h の中にある。 SIGABRT　異常終了 SIGFPE　算術エラー SIGILL　不正命令 SIGINT　割り込み（MS-DOS では Ctrl+C） SIGSEGV　メモリへの不正アクセス SIGTERM　終了要求	

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

返却値：擬似乱数を返す

説　明：0～RAND_MAX までの整数の擬似乱数を返す。RAND_MAX は stdlib.h の中で定義されている。16 ビットの処理系では 32767（0x7FFF）であることが多い。32 ビットの処理系では 2147483647（0x7FFFFFFF）であることが多い。

```
/* モンテカルロ法で円周率の値を求める */
#include <stdio.h>
#include <stdlib.h>

#define N (1024 * 1024) /* プロットする点の数 */

int main(void)
{
     double scale = 1.0 / RAND_MAX;
     double pi;
     long n = 0;
     long i;
     for (i = 0; i < N; i++) {
       double x = scale * rand();
       double y = scale * rand();
       double r = x * x + y * y;
       if (r < 1.0) {
         ++n;
       }
     }
     pi = (double)n / N * 4.0;
     printf("pi = %f¥n", pi);
     return 0;
}

【実行結果（例）】
pi = 3.142765
```

realloc — `#include <stdlib.h> void *realloc(void *ptr, size_t newsize);`

返却値：正常時：再確保されたメモリブロックへのポインタ
エラー時：NULL

説　明：ptr で示されるすでにメモリ確保されたブロックのサイズを newsize に変更して再割り当てする。古いメモリブロックにあったデータは，可能な限り新しいメモリブロックにコピーされる。

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

int main(void)
{
     char *ptA, *ptB, *ptc;
     ptA = (char *)malloc(256);

     ptB = (char *)realloc(ptA, 1024);
     /* 再確保 領域増加*/

     ptC = (char *)realloc(ptB, 128);
     /* 再確保 領域減少 */
     return 0;
}
```

remove — `#include <stdio.h> int remove(const char *filename);`

返却値：正常時：0　エラー時：非 0

説　明：filename で示すファイルを削除する。

```
if(remove("temp.tmp") != 0)
     fputs("テンポラリファイルを削除できません¥n",
          stderr);
```

rename — `#include <stdio.h> int rename(const char *oldname, const char *newname);`

返却値：正常時：0　エラー時：非 0

説　明：ファイル名を oldname から newname に変更する。

```
rename("20021201.dat", "20021202.dat");
```

rewind — `#include <stdio.h> void rewind(FILE *fp);`

返却値：なし

説　明：ファイル fp のファイル位置指示子を先頭(0)にし，エラー指示子をクリアする。
ファイル位置指示子移動に関しては，fseek(fp，0L，SEEK_SET)；と同じ。

```
【用例 1】
rewind(fp);   /* ファイルポインタを先頭へ戻す */
puts("もう一度読み直します");
while ((c = getc(fp)) != EOF)
     putchar(c);

【用例 2】
rewind(stdin);    /* キー入力クリア */
c = getchar();    /* 1 文字入力 */
```

scanf — #include <stdio.h> int scanf(const char *format, ...);

返却値	正常時：入力データの個数
	入力終了／エラー時：EOF
		int idt;
		long ldt;
		float fdt;
		double ddt;
		char sdt[80];
		
		scanf("%d", &idt);
		scanf("%ld", &ldt);
		scanf("%f", &fdt);
		scanf("%lf", &ddt);
		scanf("%s", sdt);

説　明	stdin から書式 format に従って対応する引数にデータを読み込む。	

setbuf — #include <stdio.h> void setbuf(FILE *fp, char *buf);

返却値	なし	標準出力にバッファを割り当てる。 char buf[4096]; setbuf(stdout, buf);
説　明	ファイル fp の入出力バッファを buf に指定する。通常，入出力バッファはファイルオープン時に自動的に割り当てられる。setbuf はユーザーが用意した領域を入出力バッファとして使う。setbuf は自動設定バッファ領域が不満な場合に使用するものである。用意する buf のサイズは BUFSIZ で確保する。BUFSIZ は stdio.h の中で定義されている適切な値である。buf の代わりに NULL を使うとバッファリングなしを指定する。setbuf 関数は setvbuf 関数の簡略形である。	

setjmp — #include <setjmp.h> int setjmp(jmp_buf env);

返却値	直接呼び出しのとき：0
	longjmp からの呼び出しのとき：longjmp のリターンコード
	longjmp のリターンコードが 0 のとき：1
		#include <stdio.h>
		#include <setjmp.h>
		jmp_buf buf_env;
		void func(void);
		
		int main(void)
		{
		　　int r;
		　　…
		　　r = setjmp(buf_env);
		　　…
		}
		
		void func()
		{
		　　…
		　　longjmp(buf_env,0);
		　　…
		}

説　明	現在の環境情報を env に設定し，longjmp に備える。longjmp 関数が実行されると，この位置に大域ジャンプしてくる。jmp_buf 型は setjmp.h の中で定義されている。	

setlocale — #include <locale.h> char *setlocale(int category, const char *locale);

返却値	正常時：新しい地域のカテゴリに関する情報を含む文字列へのポインタ　エラー時：NULL	【用例 1】
		setlocale(LC_ALL, "C");
		
		setlocale(LC_ALL, "jpn");
		setlocale(LC_ALL, "usa");
		setlocale(LC_MONETARY, "usa");
		setlocale(LC_ALL, "");
		
		【用例 2】
		localeconv 関数を参照。

説　明	ロケール（地域性）を設定する。ロケールとは国や文化や言語に関する約束ごと全般のことで，通貨記号（¥や$）や日付／時刻の表現方法などがこれに属する。category で指定するカテゴリ（設定，検索したい部分）に，locale で指定する地域情報（通貨記号や数値表示形式，時刻表示形式など）を設定する。文字操作や比較関数（例：strcoll）などに国状を反映させるようにする。引数 locale が"c"のときは，C に関する最小限の共通表現方法を規定する。引数 locale が""のときは，処理系に依存する地域性をもつ設定になる。処理系によって設定が異なり，たとえば Microsoft Visual C++では"jpn"や"usa"を指定できる。	

setvbuf `#include <stdio.h> int setvbuf(FILE *fp, char *buf, int mode, size_t size);`

返却値：正常時：0　エラー時：非 0

説　明：ファイル fp の入出力バッファを buf に指定する。そのとき次のモードを設定できる。

_IOBUF	フルバッファリング
_IOLBF	行バッファリング
_IONBF	バッファリングしない

行バッファリングとは，復改文字がきたときとバッファ満杯時に掃きだすモードである。バッファのサイズを size で指定する。buf の代わりに NULL を使うとバッファリングなしを指定する。setvbuf はバッファのモードとサイズを指定できる setbuf である。（setbuf 参照。）

```
FILE *fp;
char buf[128];

setvbuf(fp, buf, _IOLBF, sizeof(buf));
バッファしないとき，
setvbuf(fp, NULL, _IONBF, 0);
または setbuf(fp, NULL)
```

signal `#include <signal.h> void (*signal(int sig, void(*handler)(int)))(int);`

返却値：正常時：handler の前の値　エラー時：SIG_ERR

説　明：シグナル処理とは，外部機器からの割り込みやプログラム中での非同期割り込みなどを処理するものである。そのシグナル処理の種類は sig で指定される。sig シグナルは raise によって発生させることもできる。そのシグナルに対して適切な処理をする関数が必要で，これをシグナルハンドラといい，handler で指定する。signal 関数は，シグナルが（割り込みが）検出されたときにどういう処置をすればいいかをあらかじめシステム環境に知らせる役目をもつ。handler として SIG_DFL を指定すると標準の処理を行う。handler として SIG_IGN を指定するとそのシグナルは無視する。それ以外では指定された関数が実行される。

Ctrl+C を無効にする

```
#indlude <stdio.h>
#include <signal.h>
void ctrl_C(int n);

int main(void)
{
    int i;
    signal(SIGINT, ctrl_C);
    while(!0) {
        putchar('c');
    }
    return 0;
}

void ctrl_C(int n)
{
    signal(SIGINT, ctrl_C);
}
```

sin `#include <math.h> double sin(double x);`

返却値：計算結果

説　明：サイン値を返す。

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

int main(void)
{
    printf("sin(pi/6) = %f¥n", sin(3.141593/6));
    return 0;
}

【実行結果】
sin(pi/6) = 0.500000
```

sinh `#include <math.h> double sinh(double x);`

返却値：計算結果

説　明：ハイパボリックサイン値を返す。

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

int main(void)
{
    printf("sinh(0) = %f¥n", sinh(0));
    return 0;
}

【実行結果】
sin(0) = 0.000000
```

sprintf — `#include <stdio.h> int sprintf(char *str, const char *format,...);`

返却値	正常時：str に書き出した文字数（ナル文字は除く） エラー時：負の値
説　明	対応する引数の内容を format で指定する書式に従って文字配列 str に出力する。

```
char buf[128];
sprintf(buf, "データ%d", 123);
puts(buf);
```

sqrt — `#include <math.h> double sqrt(double x);`

返却値	計算結果
説　明	x の平方根を計算する。

```
double da;
da = sqrt(3.5);
```

srand — `#include <stdlib.h> void srand(unsigned seed);`

返却値	なし
説　明	擬似乱数の発生系列を seed によって変更する。

```
srand((unsigned)time(NULL));
時間の数値を seed とすることで，発生系列を変える。
```

sscanf — `#include <stdio.h> int sscanf(const char *str, const char *format, ...);`

返却値	正常時：入力データの個数　エラー時：EOF
説　明	文字列 str から書式 format に従って対応する引数にデータを読み込む。

```
char buf[256] = "123";
int ia;
sscanf(buf, "%d", &ia);
```

strcat — `#include <string.h> char *strcat(char *s1, const char *s2);`

返却値	連結後の s1 を返す
説　明	文字配列 s1 の後ろに文字列 s2 を連結する。

```
char buf[10] = "123";
strcat(buf, "456");
```

strchr — `#include <string.h> char *strchr(const char *str, int ch);`

返却値	見つかったとき：その文字へのポインタ 見つからないとき：NULL
説　明	文字列 str の先頭から文字 ch を探しその位置を返す。

```
char *ps;
char str[] = "abcdefg";
ps=strchr(str, 'd');

printf("%c¥n", *ps);
```

strcoll — `#include <string.h> int strcoll(const char *s1, const char *s2);`

返却値：比較結果により以下の値を返す

s1 > s2	正
s1 = s2	0
s1 < s2	負

説　明：文字列 s1 と文字列 s2 の内容を，setlocale で設定した地域仕様の情報を使って比較する。その国の文字セットが ASCII 順番で不都合のあるときに用いる。不都合のないときは strcmp を使えばよい。strcoll は STRingCOLLate（文字列照合）である。

```
if (strcoll(s1, s2) == 0) {
    printf("%s と%s は一致してます¥n", s1, s2);
}
```

strcmp — `#include <string.h> int strcmp(const char *s1, const char *s2);`

返却値：比較結果により以下の値を返す

s1 > s2	正
s1 = s2	0
s1 < s2	負

説　明：文字列 s1 と文字列 s2 の内容を辞書順に比較する。

```
char buf[] = "12345";
if (strcmp(buf, "23") == 0) {
    puts("同じ");
} else {
    puts("違う");
}
```

strcpy — `#include <string.h> char *strcpy(char *s1, const char *s2);`

返却値	s1 を返す
説　明	文字列 s1 に文字列 s2 をコピーする。

```
char buf[128];
strcpy(buf, "string copy");
```

strcspn — `#include <string.h> size_t strcspn(const char *s1, const char *s2);`

返却値	先頭からの文字数
説　明	文字列 s1 のうち，文字列 s2 に含まれるどれかの文字が最初に現れるまでの文字数を返す。言い換えると，文字列 s2 に含まれない文字だけで構成される文字列 s1 の先頭からの連続文字数を返す。

```
i = strcspn("137xyz3a567", "abcde");
printf("%d¥n", i);
```

strerror — `#include <string.h> char *strerror(int errnbr);`

返却値	システムのもっているエラーメッセージへのポインタ
説　明	errnbr に対応した処理系定義のエラーメッセージへのポインタを返す。メッセージの内容は処理系により異なる。

```
for(i = 0; i <= 10; i++)
    printf("%d:%s¥n", i, strerror(i));
【実行結果】（ある処理系の場合）
0:No error
  …
```

strftime	#include <time.h> size_t strftime(char *s, size_t maxsize, const char *format, const struct tm *timeptr);	

返却値	正常時：結果の文字数を返す
	エラー時：0
		#include <stdio.h>
		#include <time.h>
		int main(void)
		{
		time_t now;
		struct tm *local_time;
		char ss[80];
		time(&now); /* 暦時間取得 */
		local_time = localtime(&now): /* 現地時間に変換 */
		printf("%s", asctime(local_time)); /* 今日の日付時刻を表示 */
		strftime(ss, 80, "[%a]", local_time);puts(ss); /*曜日略称 */
		strftime(ss, 80, "[%A]", local_time);puts(ss); /* 曜日 */
		strftime(ss, 80, "[%b]", local_time);puts(ss); /* 月略称 */
		strftime(ss, 80, "[%B]", local_time);puts(ss); /* 月 */
		strftime(ss, 80, "[%c]", local_time);puts(ss); /* 地域仕様日付時刻 */
		strftime(ss, 80, "[%d]", local_time);puts(ss); /* 日（1-31） */
		strftime(ss, 80, "[%H]", local_time);puts(ss); /* 24 時間表記の時間（0-23） */
		strftime(ss, 80, "[%I]", local_time);puts(ss); /* 12 時間表記の時間（1-12） */
		strftime(ss, 80, "[%j]", local_time);puts(ss); /* 1 月 1 日からの日数(1-366) */
		strftime(ss, 80, "[%m]", local_time);puts(ss); /* 月（1-12） */
		strftime(ss, 80, "[%M]", local_time);puts(ss); /* 分（0-59） */
		strftime(ss, 80, "[%p]", local_time);puts(ss); /* 午前午後 */
		strftime(ss, 80, "[%S]", local_time);puts(ss); /* 秒（0-59） */
		strftime(ss, 80, "[%U]", local_time);puts(ss); /* 通算週（日曜基準，0-53） */
		strftime(ss, 80, "[%w]", local_time);puts(ss); /* 曜日（日曜が 0） */
		strftime(ss, 80, "[%W]", local_time);puts(ss); /* 通算週（月曜基準，0-53） */
		strftime(ss, 80, "[%x]", local_time);puts(ss); /* 地域表現日付 */
		strftime(ss, 80, "[%X]", local_time);puts(ss); /* 地域表現時刻 */
		strftime(ss, 80, "[%y]", local_time);puts(ss); /* 下 2 桁西暦（00-99） */
		strftime(ss, 80, "[%Y]", local_time);puts(ss); /* 西暦 */
		strftime(ss, 80, "[%Z]", local_time);puts(ss); /* 時間帯名（不明のときはなし） */
		strftime(ss, 80, "[%%]", local_time);puts(ss); /* %自身 */
		return 0;
		}
		
		【実行結果】
		Sun Jun 07 20:48:02 1998
		[Sun]
		[Sunday]
		[Jun]
		[June]
		[06/07/98 20:48:02]
		[07]
		[20]
		[08]
		[158]
		[06]
		[48]
		[PM]
		[02]
		[23]
		[0]
		[22]
		[06/07/98]
		[20:48:02]
		[98]
		[1998]
		[]
		[%]

説　明	時刻と日付を地域の表示形式にする。timeptr が指す tm 構造体内容を format で示した内容に従って変換し，maxsize で示す文字数以内で s に格納する。	

strlen `#include <string.h> size_t strlen(const char *str);`

返却値：文字列の長さを返す

説　明：文字列 str の長さを返す。ナル文字'¥0'は数えない。

```
int in;
in = strlen("XYZ");
```

strncat `#include <string.h> char *strncat(char *s1, cost char *s2, size_t n);`

返却値：s1 を返す

説　明：文字配列 s1 の後ろに文字列 s2 の先頭 n 文字を連結する。

```
char buf1[64] = "123";
char buf2[64] = "34567";
strncat(buf1, buf2, 2);
puts(buf1);
```

strncmp `#include <string.h> int strncmp(const char *s1, const char *s2, size_t n);`

返却値：比較結果により以下の値を返す

s1 > s2	正
s1 = s2	0
s1 < s2	負

説　明：文字列 s1 と文字列 s2 の先頭 n 文字同士を辞書順に比較する。

```
if (strncmp(str, "main", 4) == 0) {
    printf("%s は main から始まります¥n", str);
}
```

strncpy `#include <string.h> char *strncpy(char *s1, const char *s2, size_t n);`

返却値：s1 を返す

説　明：文字配列 s1 に文字列 s2 の先頭 n 文字をコピーする。ナル文字の自動付加は行われないので注意。s2 の長さが n より小さいときは，n に達するまで s1 にナル文字（'¥0'）を埋める。

```
char buf1[64] = "123";
char buf2[64] = "45678";
strncpy(buf1, buf2, 3);
```

strpbrk `#include <string.h> char *strpbrk(const char *s1, const char *s2);`

返却値：見つかったとき：その文字へのポインタ
見つからないとき：NULL

説　明：文字列 s1 の中で，文字列 s2 に含まれるどれかの文字が現れる位置のポインタを返す。

```
char *p;
char buf[] = "opqrstu";

p = strpbrk(buf, "abr");
printf("%s¥n", p);
```

strrchr `#include <string.h> char *strrchr(const char *str, int ch);`

返却値：見つかったとき：その文字へのポインタ
見つからないとき：NULL

説　明：文字列 str の後ろから文字 ch を探し，その位置を返す。

```
char *p;
char s[] = "C:¥¥WINDOWS¥SYSTEM32";

p = strrchr(s, '¥¥');
*p = '¥0';
printf("%s¥n", p);
```

strspn `#include <string.h> size_t strspn(const char *s1, const char *s2);`

返却値：先頭からの文字数

説　明：文字列 s1 の中で文字列 s2 に含まれない文字が最初に現れるまでの文字数を返す。言い換えると，文字列 s2 に含まれる文字だけで構成される文字列 s1 の先頭からの連続文字数を返す。
spn は span（間隔）のこと。

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

int main(void)
{
    const char *text =
      "This is a pen. That is a eraser.";
    const char *charcter =
      "abcdefghijklmnopqrstuvwxyz"
      "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      " ";
    int n = strspn(text, charcter);
    char result[256];
    memset(result, '¥0', sizeof(result));
    strncpy(result, text, n);
    puts(result);
    return 0;
}
```

【実行結果】

```
This is a pen
```

strstr `#include <string.h> char *strstr(const char *s1, const char *s2);`

返却値
見つかったとき：その文字へのポインタ
見つからないとき：NULL

説　明
文字列 s1 の先頭から文字列 s2 を探し，その位置のポインタを返す。strchr との違いは第 2 引数が文字列である点である。

```
char *p1;
char buf[] = "string";
p1 = strstr(buf, "ing");
puts(p1);
```

strtod `#include <stdlib.h> double strtod(const char *str, char **endptr);`

返却値
正常時：変換された double の値
答えがオーバフロー：HUGE_VAL
答えがアンダフロー：0
変換が行われないとき：0

説　明
文字列 str を double 型の値に変換する。変換されなかった末尾の文字列のポインタが endptr に入る。endptr が NULL のときは，この末尾文字列の処理は行われない。変換文字列には先行空白，+-符号，eE 指数を含めることができる。HUGE_VAL の値は math.h の中で定義されている。atof(s)は，strtod(s, (char **)NULL)と同じである。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main(void)
{
    char *str;
    double da;

    da = strtod("3.14ZD", &str);
    printf("%f¥n", da);
    return 0;
}
```

strtok `#include <string.h> char *strtok(char *s1, const char *s2);`

返却値
見つかったとき：そのトークンへのポインタ
見つからないとき：NULL

説　明
文字列 s2 の中の各文字を分離記号として，文字列 s1 の中からトークンを切りだす。最初にトークンを切りだししたい文字列を s1 に入れて，strtok を実行する。この文字列は strtok 関数内部に保存され，最初のトークンへのポインタが返される。2 回目以降は s1 に NULL を指定して strtok を呼び出す。返却値が NULL になるまで strtok を実行すると，すべてのトークンを得ることができる。

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

int main(void)
{
    char s[] = "123,456.789 012/,.:345:678";
    char *p;

    p = strtok(s, "/: ,.");
    printf("%s¥n", p);
    while((p = strtok(NULL, "/: ,.")) != NULL)
      printf("%s¥n", p);
    return 0;
}
```

【実行結果】

```
123
456
789
012
345
678
```

<table>
<tr><th>strtol</th><td colspan="2">#include <stdlib.h> long strtol(const char *str, char **ndptr, int base);</td></tr>
<tr><td>返却値</td><td>正常時：変換された long の値
答えがオーバフロー：結果の符合により，long 型の最大値である LONG_MAX または最小値である LONG_MIN
答えがアンダフロー：0
変換が行われないとき：0</td><td rowspan="2">#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{

char *endp;
long la;

la = strtol(” 1234567ABC” , &endp, 10);
printf("la=%ld endp=%s¥n", la, endp);
return 0;
}

【実行結果】
la=1234567 endp=ABC</td></tr>
<tr><td>説　明</td><td>文字列 str を long 型の値に変換する。変換されなかった末尾の文字列のポインタが endptr に入る。endptr が NULL のときは，この末尾文字列の処理は行われない。変換文字列には先行空白，+-符号，8 進数を示す 0，16 進数を示す 0x, 0X を含めることができる。base は 2~32 の値をとり，それぞれ基数となる（16 なら 16 進数となる）。base が 0 のときは C の定数表現に従う。すなわち，0 が付けば 8 進数，0x，0X が付けば 16 進数，それ以外は 10 進数となる。base が 16 のときは 0x，0X 表現を用いてよい。LONG_MAX 及び LONG_MIN は，limits.h の中で定義されている。atoi(s)は(int)strtol(s，(char **)NULL,10)と同じである。atol(s)は，strtol(s，(char **)NULL，10)と同じである。</td></tr>
<tr><th>strtoul</th><td colspan="2">#include <stdlib.h> unsigned long strtoul(const char *str, char **endptr, int base);</td></tr>
<tr><td>返却値</td><td>正常時：変換された unsigned long の値
答えがオーバフロー:結果の符合により long 型の最大値である LONG_MAX または最小値である LONG_MIN
答えがアンダフロー：0
変換が行われないとき：0</td><td rowspan="2">#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int main(void)
{
char ss[] = "1234567890XYZ";
char *endp;
unsigned long d;
d = strtoul(ss, &endp, 10);
printf("d=%lu endp=%s¥n", d, endp);
return 0;
}

【実行結果】
d=1234567890 endp=XYZ</td></tr>
<tr><td>説　明</td><td>文字列 str を unsigned long 型の値に変換する。その他の変換規則は strtol と同じ。strtol 参照。</td></tr>
<tr><th>strxfrm</th><td colspan="2">#include <string.h> size_t strxfrm(char *s1, const char *s2, size_t n);</td></tr>
<tr><td>返却値</td><td>変換された文字数</td><td rowspan="2">文字順列に問題があるとき次の二つの結果は同じになる。
(1)if(strcoll(s1, s2) == 0){
（何かの処理）
}
(2)strxfrm(s11, s1, 80);
strxfrm(s22, s2, 80);
if(strcmp(s11, s22) == 0){
（何かの処理）
}</td></tr>
<tr><td>説　明</td><td>文字列 s2 を地域仕様に基づいて先頭から最大 n バイトを変換し，s1 に入れる。その国の文字セットが ASCII 順番と辞書順で不都合のあるときがある。たとえば，ある国では文字コードは'e'，'ë'の順だが，辞書順は'ë'，'e'となる。このような場合は文字列比較に strcmp ではなく，strcoll を使わなくてはならない。しかし，strxfrm で文字列変形すれば strcmp を使用できる。日本ではこの関数を使う必要はない。</td></tr>
</table>

system	#include <stdlib.h> int system(const char *cmnd);	
返却値	cmnd が NULL のときの返却値 　コマンドプロセッサが存在するとき：非 0 　存在しないとき：0 cmnd が文字列のときの返却値は処理系に依存する 　通常は，成功時：0　エラー時：-1	system("type autoexec.bat");
説　明	コマンドプロセッサを呼び出し，cmnd で指定したコマンドを実行する。system(NULL);と書いたときはコマンドプロセッサの存在をチェックする。	
tan	#include <math.h> double tan(double x);	

返却値	計算結果	#include <stdio.h>
		#include <math.h>
		
		int main(void)
		{
		　　printf("tan(pi/4) = %f¥n", tan(3.141593/4));
		　　return 0;
		}
		
		【実行結果】
		tan(pi/4) = 1.000000

説　明	タンジェント値を返す。	
tanh	#include <math.h> double tanh(double x);	

返却値	計算結果	#include <stdio.h>
		#include <math.h>
		
		int main(void)
		{
		　　printf("tanh(0) = %f¥n", tanh(0));
		　　return 0;
		}
		
		【実行結果】
		tanh(0) = 0.000000

説　明	ハイパボリックタンジェント値を返す。	
time	#include <time.h> time_t time(time_t *timeptr);	
返却値	正常時：現在の暦時刻 エラー時：(time_t)(-1)	time_t now; time(&now);
説　明	現在の暦時刻を timeptr に入れる。timeptr に入る値と，この関数の返却値は同じものなので， 　t=time(NULL); のように使ってもよい。time_t 型については“処理系で定義される型”を参照。	
tmpfile	#include <stdio.h> FILE *tmpfile(void);	

返却値	正常時：FILE へのポインタ　エラー時：NULL	#include <stdio.h>
		int main(void)
		{
		　　FILE *tmp = tmpfile();
		　　char buf[256];
		　　fputs("temporary file test!!", tmp);
		　　rewind(tmp);
		　　fgets(buf, 256, tmp);
		　　puts(buf);
		　　return 0;
		}
		
		【実行結果】
		temporary file test!!

説　明	一時的にデータを置くためのファイルを作る。ファイルは wb+（バイナリ更新モード）でオープンされる。このファイルはファイルクローズかプログラム終了で自動的に削除される。	

<table>
<tr><th>tmpnam</th><td colspan="2">#include <stdio.h> char *tmpnam(char *name);</td></tr>
<tr><td>返却値</td><td>name が NULL のときは内部的に生成した名前へのポインタ。name が NULL でないときはポインタ name。</td><td rowspan="2">【用例 1】名前を作る。
#include <stdio.h>
int main(void)
{
FILE *tmp;
char myname[L_tmpnam];
tmpnam(myname);
puts(myname);
return 0;
}

【実行結果】
TMP1.$$$

【用例 2】
名前用変数を使用せずに一時ファイルをオープンする。
FILE *fp = fopen(tmpnam(NULL), "wb+");</td></tr>
<tr><td>説　明</td><td>一時的にデータを置くためのファイルの名前を作り，name に入れる。この名前はディスク上の既存ファイル名と衝突しない。name が NULL のときは生成した名前を内部的なメモリ領域に置き，そのポインタを返す。name のサイズは少なくとも L_tmpnam バイト必要である。この値は stdio.h の中で定義されている。tmpnam は名前だけを作る関数である。そのファイルを実際にオープン／クローズするのは fopen，fclose による。</td></tr>
<tr><th>tolower</th><td colspan="2">#include <ctype.h> int tolower(int c);</td></tr>
<tr><td>返却値</td><td>アルファベットのとき：小文字
それ以外のとき：そのままの値</td><td rowspan="2">int ca;
ca = tolower('A');</td></tr>
<tr><td>説　明</td><td>文字 c が大文字なら小文字に変換して返す。</td></tr>
<tr><th>toupper</th><td colspan="2">#include <ctype.h> int toupper(int c);</td></tr>
<tr><td>返却値</td><td>アルファベットのとき：大文字
それ以外のとき：そのままの値</td><td rowspan="2">int ca;
ca = toupper('x');</td></tr>
<tr><td>説　明</td><td>文字 c が小文字なら大文字に変換して返す。</td></tr>
<tr><th>ungetc</th><td colspan="2">#include <stdio.h> int ungetc(int c, FILE *fp);</td></tr>
<tr><td>返却値</td><td>正常時：文字 c　エラー時：EOF</td><td rowspan="2">ファイルの先頭の文字が'a'のときだけファイルを表示する。

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

int main(void)
{
FILE *fp;
int ch;

fp = fopen("file.txt", "r");
ch = fgetc(fp);
if(ch == 'a') {
ungetc(ch, fp);
} else {
exit(1);
}
while((ch = fgetc(fp)) != EOF) {
putchar(ch);
}
return 0;
}</td></tr>
<tr><td>説　明</td><td>ファイル fp に文字 c を戻す。戻された文字は，まだ読んでいなかったものとして扱われる。ungetc は，先読み処理した文字を元の状態に戻すのに使われる。</td></tr>
<tr><th>vfprintf</th><td colspan="2">#include <stdio.h> #include <stdarg.h> int vfprintf(FILE *fp, const char *format, va_list arg);</td></tr>
<tr><td>返却値</td><td>正常時：出力した文字数　エラー時：負の値</td><td rowspan="2">vprintf()を参照。</td></tr>
<tr><td>説　明</td><td>arg を可変個の引数リストとして fprintf を行う。</td></tr>
</table>

vprintf	#include <stdio.h> #include <stdarg.h> int vprintf(const char *format, va_list arg);	

返却値	正常時：出力した文字数　エラー時：負の値	#include <stdio.h>
		#include <stdarg.h>
		int myprint(char *fmt, ...);
		
		int main(void)
		{
		　　myprint("%d + %d = %d¥n", 32, 15, (32+15));
		　　return 0;
		}
		
		int myprint(char *fmt, ...)
		{
		　　int ret;
		　　va_list pt;
		
		　　va_start(pt, fmt);
		　　ret = vprintf(fmt, pt);
		　　va_end(pt);
		　　return (ret);
		}

説　明	arg を可変個の引数リストとして printf を行う。可変個の引数リストを va_list 型 arg に設定して関数を呼び出す。arg は引数リストの先頭アドレスを示す。va_list は stdarg.h の中で定義されている。(通常 void *型)。arg は使用前に va_start マクロで初期化しなければならない。arg は使用後に va_end マクロで後処理しなければならない。ただし，現実に後処理する仕事はないのが普通である。そのとき，va_end は無効果のマクロとして定義される。この場合，va_end は va_start との対象性をとるための役目をもつだけとなる。	
vsprintf	#include <stdio.h> #include <stdarg.h> int vsprintf(char *str, const char *format, va_list arg);	
返却値	正常時：書き出した文字数（ナル文字を除く） エラー時：負の値	vprintf を参照。
説　明	arg を可変個の引数リストとして sprintf を行う。	
wcstombs	#include <stdlib.h> size_t wcstombs(char *mbstr, const wchar_t *wcstr, size_t count);	

返却値	マルチバイト文字列に変換したバイト数を返す	#include <stdio.h>
		#include <stdlib.h>
		#include <locale.h>
		
		int main(void)
		{
		　　int i;
		　　char pmbbuf[MB_CUR_MAX+1];
		　　wchar_t *pwchello = L"Hello, world.";
		
		　　setlocale(LC_ALL, "jpn");
		　　printf("ワイド文字列の変換:¥n");
		　　i = wcstombs(pmbbuf, pwchello, MB_CUR_MAX);
		　　printf("¥t 変換された文字数: %u¥n", i);
		　　printf("¥t マルチバイト文字: %s¥n¥n", pmbbuf);
		　　return 0;
		}

説　明	ワイド文字列を対応するマルチバイト文字列に変換する。通常，ワイド文字列をマルチバイト文字列に変換したときに必要になるバイト数はわからない（1 バイトに変換されるものと，2 バイトに変換されるものとがある）。	
wctomb	#include <stdlib.h> int wctomb(char *mbchar, wchar_t wchar);	

返却値	ワイド文字の長さを返す。現在のロケールで変換できない場合は-1 とする。	#include <stdio.h>
		#include <stdlib.h>
		
		int main(void)
		{
		　　int i;
		　　wchar_t wc = L'a';
		　　char pmb[MB_CUR_MAX + 1];
		
		　　printf("ワイド文字の変換:¥n");
		　　i = wctomb(pmb, wc);
		　　printf("¥t 変換された文字数: %d¥n", i);
		　　printf("¥t マルチバイト文字: %.1s¥n¥n", pmb);
		　　return 0;
		}

説　明	wchar で示すワイド文字を対応するマルチバイト文字に変換する。	

３．書式指定文字列

（１）　出力書式指定文字列

各変換文字列は，'%'で始まり変換文字で終わる。'%'と変換文字の間には次のものをこの順で記述してもよい。変換文字列が以下に合致しない場合，動作は不定である。

1. 仕様を変更するためのフラグ（順序は任意）。
   - `-`　　：フィールド内での左揃え
   - `+`　　：数値の前に符号を付ける
   - 間隔文字　：最初の文字が符号でなければ，間隔文字を前に付ける
   - `0`　　：数値の場合，フィールド一杯に左から 0 をつめる
   - `#`　　：別の出力形式の指定
2. 最小フィールド幅を指定する数。少なくともこの幅で出力される。
3. ピリオド。フィールド幅と精度を分離。
4. 精度。s の場合は出力すべき最大文字数。e，E，f では小数部の桁数。g，G は実数の場合は，有効数字の数。整数の場合は，出力すべき最小桁数。
5. 長さ修飾
   - `h`　　：short または unsigned short
   - `l`　　：long または unsigned long
   - `L`　　：long double

幅，もしくは精度（両方でもよい）は'*'でもよい。このときは次の引数（int）を計算することで得ることになる。変換文字は次のようになっている。

変換文字	対象	内容
d，i	int	符号付き 10 進数
o	int	8 進数
x，X	int	16 進数
u	int	符号なし 10 進数
c	int	unsigned char に変換された後の 1 文字
s	文字列	char 型配列
f	double	固定小数点表示
e，E	double	浮動小数点表示
g，G	double	e（E）と f を適宜切り替える
p	void *	ポインタとして出力，処理系に依存する
n	int *	これまで書き出された文字数が引数に書き出される
%		%を出力

（２）　入力書式指定文字列

書式文字列には次のものが書ける。

●間隔（スペース）文字，タブ文字。これらは何の機能も果たさないので，可読性を増すために入れることができる。

●%でない文字。入力ストリーム中の次の非空白類文字とマッチすべきもの。

●変換仕様

1. %変換仕様の始まりを表す。
2. *代入抑止文字。この文字があると変換したものを変数に格納しない。
3. 最大フィールド幅を指定する数（省略可）。
4. 変数の型を示す h，l，L。省略した場合はデフォルトになる。
   ただし，l は以下のように解釈される。
   これに続く変換文字が d，i，o，u，x のときは，出力書式指定文字列の長さ修飾。これに続く変換文字が e，f，g の時は double。
5. 変換文字

入力ストリームにおいて空白類文字は，間隔，タブ，改行，復帰，垂直タブ，改ページである。

変換文字	対象	内容
d	int *	10 進数
i	int *	整数。8 進（先頭に 0），16 進（先頭に 0x，0X）も可
o	int * unsigned int *	8 進数整数。先頭の 0 は任意
u	unsigned int *	符号なし 10 進数
x，X	int * unsigned int *	16 進数。先頭に入力する 0x，0X はどちらでもよく，また任意でもある。x，X と 0x，0X は無関係
c	char *	文字
s	char *	非空白類文字の文字列
e，E，f，g，G	float *	浮動小数点数。どの変換文字でも同じ。指数形式も可
p	void *	ポインタの値
n	int *	これまでに読み込まれた文字数
%	なし	代入は何も行われない
[…]	char *	操作文字集合変換。括弧の間にある入力文字の，空でない最長の文字列とマッチ
[^…]	char *	操作文字集合変換。括弧の間にない入力文字の，空でない最長の文字列とマッチ

平成 27 年 4 月 8 日　作成