

C言語講座

第6回

システム関数

今回は、C言語に備わっている便利な関数群について、有用だと思われるものをいくつかピックアップして教えていこうと思う

・・・ので、ライブラリ関数を扱うために必要な、“ヘッダをインクルードする”ということを先に学んでいこう。

ヘッダ(header)

その名の通り...と言っているのか分からないが、たいていはソースコードの上部、つまり頭に書きたいプログラムを記述したファイル。

具体的には、

プロトタイプ宣言、構造体・共有体の定義、
enum定義、グローバル関数の宣言、
インライン関数の宣言・定義

などなど。

インクルード(include)

ヘッダファイルで宣言したものを取り込む。

include自体には“含む”という意味があり、
まあ、ここでの意味的には“読み込む”程度に
思ってくれれば良い。

○構文

```
#include "ファイル名"
```

○プログラミング実践

変数を入れ替える

test.h

```
#include "stdio.h"
void swap(int *x, int *y);
void print_ab(int a, int b);
```

```
#include "test.h"
void main(void) {
    int a = 3, b = 9;
    print_ab(a, b);
    swap(&a, &b);
    print_ab(a, b);
}
void swap(int* x, int* y) {
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
void print_ab(int a, int b) {
    printf("a=%d, b=%d\n", a,b);
}
```

イメージ的には
こうなる

```
#include "stdio.h"
void swap(int *x, int *y);
void print_ab(int a, int b);

void main(void) {
    int a = 3, b = 9;
    print_ab(a, b);
    swap(&a, &b);
    print_ab(a, b);
}

void swap(int* x, int* y) {
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

void print_ab(int a, int b) {
    printf("a=%d, b=%d\n", a,b);
}
```

ちなみに。

“#”がつくキーワードはプリプロセッサと呼ぶ、
コンパイラに対する特殊命令。

#include、#define、#ifndef、#if、#endifなど、
いろいろな種類がある。

gets	32
malloc	71,281
free	71,278
memset	53,308
str系	47~
rand	69,290
fopen	26,203

stdio.h(Standard Input/Output)

標準入出力を行うための機能が盛り込まれている。代表的なものを紹介する。

入出力:

printf, scanf, gets ,puts

ファイルアクセス:

fopen, fclose

gets, puts (get/put String)

行単位の入出力

具体例

```
#include <stdio.h>

main()
{
    char s[256];
    gets(&s);
    puts(&s);
}
```

time.h(time)

時間操作。

時間取得：

time

time()

歴時刻(システム時間)の取得。

現在の日付を取得できる。

具体例

```
#include <time.h>
void main() {
    time_t now;
    time(&now);
    printf(“%d年%月%d日”, now.tm_year + 1900,
           now.tm_mon+1, now.tm_mday);
}
```

stdlib.h(Standard library)

一般ユーティリティ。

ランダム:

srand, rand

メモリ管理:

malloc, free

ファイルアクセス:

fopen, fclose

fopen(file open)

filenameで示すパスのファイルを指定したmodeで開く。

constは定数の意。
ここでは気にしなくても良い。

○構文

```
FILE *fopen(const char *filename, const char *mode)
```

mode:

mode	read/write	シーク	ファイル存在時	ファイルが無かった
r	read	先頭	何もしない	エラーで返す
w	write	先頭	空ファイルにする	空で作成する
a	write	終末	何もしない	空で作成する
r+	read/write	先頭	何もしない	エラーで返す
w+	read/write	先頭	空ファイルにする	空で作成する
a+	read/write	終末	何もしない	空で作成する

fclose(file close)

ストリームに結合したファイルを解放する。
オープンしたファイルは必ずこれを行わなければならない。

○構文

```
int fclose(File *stream);
```

fopen, fclose (file open/close)

具体例

```
#include <stdio.h>
void main()
{
    FILE* pf;
    char s[256];
    if ((pf = fopen("text.txt", "r")) != NULL) {
        fgets(&s, sizeof(s), pf);
        printf("%s", s);
        fclose(pf);
    } else {
        printf("file not found...");
    }
}
```


srand, rand (seed random, random)

srand:乱数の初期化

rand:乱数の取得(0 ~ 3)

具体例

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void main(void) {
    inti;
    srand( (unsigned)time(NULL) );
    for(i=0;i < 10; i++){
        printf("乱数=%d\n", rand() % 100);
    }
}
```

動的メモリ領域割り当てってって？

これの反対語が静的(ry)。

例えば、`char s[1000][100];`
という配列を宣言すると、 $1000 * 100 = \text{約}100\text{K}$
byteのメモリを取得するが、これは実行時に
変更することが出来ない。足りなくなるかも知
れないし、多すぎて無駄かも知れない。

だから、

必要な時に必要なだけメモリを確保したい

malloc, free (メモリ管理)

malloc:指定バイト分、メモリ領域を確保する

free:メモリ領域を解放する

具体例

```
#include <stdlib.h>

void main()
{
    char *p;
    p = (char*)malloc(10);
    free(p);
}
```

char * p[10];
と同じ