C 言語のサンプルプログラム (5)

計算機応用　5E 2004.06.03

1．コンソール入力

コンソール(ディスプレイやキーボート)の入出力は、いろいろあります。ここでは、数値計算法を学ぶために必要なことのみ述べます。これ以上の詳細は、C 言語の解説書を読んでください。

コンソール入出力のことを、標準入出力と言う場合があります。また、後で(P.11)述べますが、ファイルの入出力先を指定するために、この標準入出力には、名前が付いています。C 言語では、これらの名前を用いて、入出力先の指定を行います。これらの関係を、以下に示します。

キーボード	→	標準入力	→	stdin
ディスプレイ	→	標準出力	→	stdout

1.1 書式付入力(scanf)

入力は、少し癖は有りますが、文字や数字が同じように入力できる scanf 関数を用います。この関数を使う上での注意は、

- データを入れる引数は、ポインターである。
- %s の書式(文字列)で文字列を読み込む場合、空白は読み込みません。空白の前まで、読み込みは終わりです。
- %c の書式(1 文字)であれば、空白を読み込みます。
- 復改文字(\n)がバッファーに残ります。これを、読み捨てないと、次の入力でこれが読み込まれます。

等です。

この書式付入力 scanf 関数の定義は、

```
int scanf(書式指定, 引数並び)
```

です。戻り値は、正常時は入力した項目数ですが、異常時は EOF を返します。

入力の文字や数値の書誌指定は、

```
% [代入抑止文字] [最大フィールド幅] [変換修飾子] 変換指定子
```

です。[ ]内は省略可能です。それぞれについては、教科書の p.328～329 に掲載されています。

1 個の scanf 関数で複数のデータを入力することが可能です。そのときのデーターの区切りは、空白、Tab、改行で行われます。特別な文字で、データを区切りたいときには、書式に書きます。printf と同じです。

いろいろありますが、数値計算で使うのは、主に、

```
%lf   倍精度実数型(double)入力
%d    10 進数整数入力
%s    文字列入力
%c    指定文字数入力。1 文字のとき、よくつかう。
```

です。

1.2 書式付入力サンプルプログラム

サンプルプログラム(char.c)

1 文字入力のプログラムです。読み込んだ文字を出力します。temp には、復改文字(\n)が代入されます。この復改文字は、通常のプログラムでは不要なので、読み捨てます。もし読み捨てないと、バッファーに残り、次の読み込みで不都合が生じます。

```
#include <stdio.h>

int main(){
   char a, temp;

   scanf("%c%c", &a, &temp);

   printf("%c\n",a);

   return(0);
}
```

サンプルプログラム(string.c)

最大 256 文字までの文字列を読み込みます。そして、読み込んだ文字を出力します。

```
#include <stdio.h>

int main(){
   char a[256], temp;

   scanf("%s%c", a, &temp);

   printf("%s\n",a);

   return(0);
}
```

サンプルプログラム(string2.c)

最大 256 文字までの文字列を、2 文字列読み込みます。そして、読み込んだ文字を出力します。

```
#include <stdio.h>

int main(){
   char a[256], b[256];
   char temp;

   scanf("%s%s%c", &a, &b, &temp);

   printf("%s\n",a);
   printf("%s\n",b);

   return(0);
}
```

サンプルプログラム(double2.c)

double 型で、2 個の数値を読み込みます。そして、読み込んだ数値を出力します。

```
#include <stdio.h>

int main(){
   double a, b;
   char temp;

   scanf("%lf%lf%c",&a,&b,&temp);

   printf("%f\n",a);
   printf("%f\n",b);

   return(0);
}
```

サンプルプログラム(integer.c)

8, 10, 16 進数の整数を読み込みます。そして、読み込んだ整数を 10 進数で出力します。

```
#include <stdio.h>

int main(){
   int a;
   char temp;

   scanf("%i%c", &a, &temp);

   printf("%d\n",a);

   return(0);
}
```

実行結果

8 進数の 7777 と, 10 進数の 4095, 16 進数の fff が同じであることが分かります。

8 進数の場合	10 進数の場合	16 進数の場合
07777	4095	0xfff
4095	4095	4095

2．コンソール出力

2.1 書式付出力(printf)

ディスプレイへの出力は、今までおなじみの printf を使います。この書式付出力 printf 関数の定義は、

```
int printf(書式指定、引数並び)
```

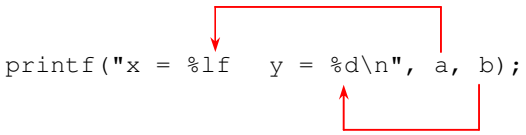
です。戻り値は、正常時は出力した文字数ですが、異常時は負の数を返します。

出力の文字や数値の書誌指定は、

```
% [フラグ] [最小フィールド幅] [精度] [変換修飾子] 変換指定子
```

です。[ ]内は省略可能ですそれぞれについては、教科書の p.321～322 のようになります。メモリーの内容を変換指定子で定められた書式に、変換し、文字出力します。

書式指定中の書式指定と変数との対応は、以下の通りです。いままで、さんざん練習してきましたので、わかっていますよね。書式指定と変数は、1 対 1 で順番通りに対応しています。



いろいろありますが、数値計算で使うのは、主に、

```
%f    浮動小数点出力　桁数が不足する場合があるの%e の方が望ましい。
%e    指数出力
%d    10 進数整数出力
%s    文字列出力
%c    文字出力。1 文字のとき、よくつかう。
```

です。

2.2 書式付出力サンプルプログラム

書式付出力の printf 関数は、さんざん使ってきたので、必要なことはほとんど説明済みです。詳細な出力指定をしたいときは、表 5〜9 を見て、自分で考えてください。

ここでは、少し変わったサンプルプログラムを示します。三角関数のグラフを書きます。

サンプルプログラム(sin.c)

最小フィールド幅*を使って、三角関数のグラフを書きます。よく FORTRAN の教科書にあるやつです。数学関数を使っているので、コンパイルにはオプション(-lm)が必要です。

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

int main(){
   int i, iy, n, column;
   double x;
   double pi=4*atan(1);

   n = 25;

   for(i=0; i <= n; i++){
      x = 2*pi/n*i;
      iy = 30*sin(x)+40;

      printf("%*c\n",iy,'*');
   }

   return(0);
}
```

サンプルプログラム(outnum.c)

整数と実数をいろいろな形式で出力します。

```
#include <stdio.h>
main()
{
  char tmp;
  int i;
  double x;

  printf("write integer number  = ");
  scanf("%d%c%", &i, &tmp);

  printf("write real number     = ");
  scanf("%lf%c%", &x, &tmp);

  printf("\n");

  printf("decimal      : %d\n",i);
  printf("octal        : %o\n",i);
  printf("hexadecimal  : %x\n",i);

  printf("\n");

  printf("float        : %f\n",x);
  printf("exponent     : %e\n",x);
  printf("g type       : %g\n",x);
}
```

実行結果

いろいろな数字を入力して、出力を試して見ましょう。

```
write integer number = 123456
write real number    = 123.456e-7

decimal       :  123456
octal         :  361100
hexadecimal   :  1e240

float         :  0.000012
exponent      :  1.234560e-05
g type        :  1.23456e-05
```

3．ファイル処理関数

ここでは、数値計算に必要なハードディスクへのデータの書き込みと、そこからの読み込みについて述べます。それ以外のデバイスもほとんど同じです。各自、必要になったら学習すればよいでしょう。

数値計算では、大量の計算結果をファイルに書き込んだり、ファイルから計算すべきデータを読み込んだりします。また、他のプログラムとのデータの受け渡しに、ファイルが使われます。例えば、計算結果をグラフにする場合、一度ファイルにデータを書き込んで、グラフを書くソフトウェアーに渡します。そうすると、グラフを書くソフトウェアーを書く手間が省けます。

ファイルへのアクセスの方法は、順番通りアクセスするシーケンシャルアクセスと、いろいろなところからアクセスするランダムアクセスがあります。数値計算で、ランダムアクセスを使うことは希なので、ここでは取り扱わないこととします。

3.1 ファイルのオープンとクローズ

ほとんどのプログラム言語では、ファイルの処理は、図 1 のようになっています。これは、約束事なので覚えてください。当然、複数個のファイルをオープンすることもできます。

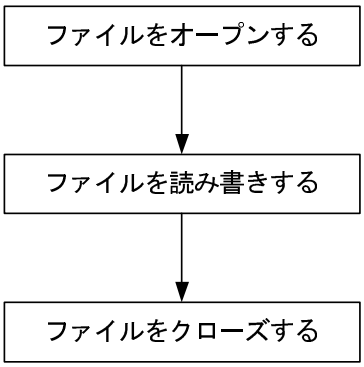


図 1 ファイル処理の流れ

これら 3 個の処理のうち、簡単なオープンとクローズについて、ここで説明します。ファイルの読み書きは、これらより少し複雑なので、次節以降で説明します。

サンプルプログラム(1)でファイルの読み書きについて、簡単に説明しましたが、ここではもう少し詳しく述べます。

C 言語では、かなり細かいファイルの処理が出来ます。そのために、ファイルの情報は、ファイルポインターと言われる構造体に記述されます。これは、stdio.h というヘッダーファイルに定義されています。すべてこのファイルポインターを使って、ファイル関係の処理は実施されます。名前の通り、これはポインターです。この値は、fopen という関数の戻り値です。

それでは、例を図 2 示して、ファイルのオープンとクローズを示します。このプログラムは、

- ファイルポインター (図 2 のプログラムの変数 fp)を定義しています。ポインターだから、*fp と変数の前にアスタリスクがついています。ファイルポインターの型は、FILE とすべて大文字で書きます。
- 次に、fopen 関数でファイルをオープンしています。この関数の戻り値が、ポインターです。
- 最後に、fclose 関数でファイルをクローズしています。

となっています。

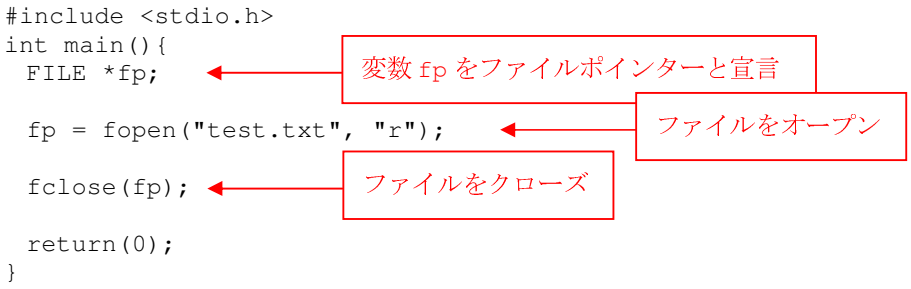


図 2 ファイルのオープンとクローズ

ファイルをオープンする関数 fopen の書式は、次の通りです。

```
FILE *open(char *filename, char *openmode)
```

戻り値は FILE 型のポインター、ファイル名を表す第一引数は char 型のポインター、オープンモードを表す第 2 引数は char 型のポインターと言う意味です。もし、オープンに失敗すると、NULL という戻り値になります。

オープンモードについては、教科書の p.382 にまとめてあります。あと、バイナリーモードというものがありますが、余り関係ないので説明は省略します。

こう言うと分かりにくいのですが、実際は、図 2 に示したように書きます。これを、真似ればよいでしょう。

ファイルをクローズする関数 fclose の書式は、簡単で、次の通りです。

```
int fclose(FILE *filepointer)
```

戻り値は、int 型で、クローズに成功すると 0、失敗すると EOF が返されます。引数は、ファイルポインターです。これも、図 2 を真似すればよいでしょう。

3.2 特別なファイル(標準入力、標準出力、標準エラー出力)

C 言語でファイルを取り扱う場合、以下のようにプログラムを作成しなくてはなりません。

① ファイルポインター用の変数を FILE 型で宣言する。
② ファイルをオープンする。
③ ファイルの読み書き
④ ファイルのクローズ

しかし、特別な 3 個のファイル(標準入力、標準出力、標準エラー出力)は、この①と②、④が不要です。この 3 個のファイルは、実行時、自動的に①③④の処理が行われます。

コンソール入力で述べたように、標準入力はキーボード、標準出力はディスプレイです。C 言語では、広くは UNIX では、キーボードやディスプレイもファイルとして扱われ、読み書きします。それどころか、すべてのデバイスがファイルとして扱われます。そうすると、シンプルな取り扱いができます。

これら、特別な 3 個のファイルについて、表 1 にまとめておきます。

表 1 標準入出力ファイル

ファイル	ファイルポインター	デバイス(通常)
標準入力	stdin	キーボード
標準出力	stdout	ディスプレイ
標準エラー出力	stderr	ディスプレイ

3.3 ファイル入出力関数

いよいよ、ファイルのデータを読み書きする、ファイル入出力関数です。これは、難しそうですが、簡単です。いままで、標準入力と標準出力に使ってきた関数、printf と scanf とほとんど同じです。コンソール入出力と言ってきたのは、標準入出力です。キーボードやディスプレイもファイルの取り扱いなので、同じ手法がハードディスクにも使えます。

まず、入力からですが、一般のファイルと標準入力の場合を並べて書くと

一般のファイル	int fscanf(ファイルポインター, 書式指定, 引数並び)
標準入力	int scanf(書式指定, 引数並び)

となります。ファイルポインターを指定する以外、すべて標準入力の場合と同じです。簡単でしょう。もちろん、fscanf 関数でファイルポインターとして stdin を指定した場合、scanf と同じ動作をします。

次に、出力ですが、これもまったく同じです。

一般のファイル	int fprintf(ファイルポインター, 書式指定, 引数並び)
標準出力	int printf(書式指定、引数並び)

これで、コンソール入出力をしつこく詳細に説明した理由がわかったでしょう。これも、fprintf 関数でファイルポインターとして stdout を指定した場合、printf 関数と同じ動作をします。

最後に標準エラー出力について述べて起きます。標準エラー出力とは、エラーが発生した場合のメッセージなどを出力先です。プログラム中で処理にエラーが発生した場合、そのメッセージの出力先に指定します。printf 関数を使うよりも、

```
fprintf(stderr, "ファイルの読み込みに失敗し増した\n")
```

とした方が、プロっぽくて良いです。エラーメッセージのみ、リダイレクトすることができプログラムの保守性が上がります。

コーヒーブレイク

リダイレクトとは、以下のように出力先を変えることです。ここの command は、実行ファイルと同じです。これは、プログラム中に書くのではなくて、UNIX のターミナルからのコマンドです。

- 標準出力を hogehoge というファイルに出力する場合

```
command > hogehoge
```

- 標準エラー出力を hogehoge というファイルにする場合

```
command 2> hogehoge
```

- 標準入力の変わりに、hogehoge というファイルを使う場合

```
command < hogehoge
```

サンプルプログラム(inout.c)

このプログラムの動作は、①標準入力から整数を読む　②それを標準出力に出力　③それをファイル one に出力 ④ファイル one を読み込む　⑤読み込んだファイル one の中身をファイル two に出力します。

ファイルへのデータの書き込みのデータ区切りに/t (tab タブ)を使っています。データの区切りとして、タブは良く使われます。これは、データをエディターで見るときに、データの並びがそろっているため、視認性がよくなるからです。

```
#include <stdio.h>
int main()
{
 int a[4], b[4];

 char tmp;
 FILE *out1, *in, *out2;

 fscanf(stdin,"%d%d%c", &a[0], &a[1], &tmp);  /* read from keybord */
 fscanf(stdin,"%d%d%c", &a[2], &a[3], &tmp);

 printf("\n");
 fprintf(stdout,"%d %d\n", a[0], a[1]);       /* write to display */
 fprintf(stdout,"%d %d\n", a[2], a[3]);

 out1 = fopen("one","w");
 fprintf(out1,"%d\t%d\n", a[0], a[1]);        /* write to file one */
 fprintf(out1,"%d\t%d\n", a[2], a[3]);
 fclose(out1);

 in = fopen("one","r");         /* read from one and write to two */
 out2 = fopen("two","w");
 fscanf(in,"%d\t%d", &b[0], &b[1]);
 fscanf(in,"%d\t%d", &b[2], &b[3]);
 fprintf(out2,"%d\t%d\n", &b[0], &b[1]);
 fprintf(out2,"%d\t%d\n", &b[2], &b[3]);
 fclose(in);
 fclose(out2);

 return(0);
}
```

実行結果

コンソールは以下のようになります。ファイル one と two についても調べましょう。

```
1 23
456 789

1 23
456 789
```