C 言語の学習
ファイル処理関数

山本昌志*

2004 年 7 月 7 日

1　本日の学習内容

本日の内容は、教科書の 18 章ファイル処理関数に付いて学習する。

18 章　ファイル処理関数

本講義のメインテーマである数値計算では、大量の数値を扱うことが多い。いちいち紙に書き写すことは不可能なので、ハードディスクに保存されるのが普通である。数値計算に限らず、現代の実験では、データの取得にはコンピューターが使われ、そのハードディスクに大量のデータが保存される。そのデータを処理して、意味のある量に加工するのである。このようなことから、ファイル処理に関わるテクニックはコンピューターを使う上で必須となっている。ファイル処理の技術を習得し、コンピューターを自在に活用して欲しい。

2　ファイル処理関数 (18 章)

2.1　ファイル処理の体験

2.1.1　ファイル出力

ごちゃごちゃとファイル処理について、説明する前に、実際にファイル処理を体験してみよう。

ファイル出力の例として、ディスプレイに出力していた”Hello World !!”をファイルに書き出す。リスト 1 のプログラムを実行させて、作成されたファイル (hello.txt) をエディター (emacs など) で内容を確認しよう。

4 行 ファイルポインターの宣言。

6 行 ファイルのオープン

8 行 ファイルへの書き込み

*独立行政法人　秋田工業高等専門学校　電気工学科

10 行 ファイルのクローズ

リスト 1: ファイル出力の例

```
1  #include <stdio.h>
2
3  int main(void){
4    FILE *fp;
5
6    fp=fopen("hello.txt","w");
7
8    fprintf(fp,"Hello World !!\n");
9
10   fclose(fp);
11
12   return 0;
13 }
```

2.1.2 ファイル入力

先ほど作成したファイル (hello.txt) の内容を読み、それを画面に出力してみよう。

5 行 ファイルポインターの宣言。

8 行 ファイルのオープン

10 行 ファイルからデータの読み込み

14 行 ファイルのクローズ

リスト 2: ファイル入力の例

```
1  #include <stdio.h>
2
3  int main(void){
4
5    FILE *fp;
6    char a[32], b[32], c[32], tmp;
7
8    fp = fopen("hello.txt", "r");
9
10   fscanf(fp, "%s%s%s%c", a, b, c, &tmp);
11
12   fclose(fp);
13
14   printf("%s %s %s\n", a, b, c);
15
16   return 0;
17 }
```

2.2 数値計算のファイル入出力

C 言語のファイル入出力は、用途に応じた処理ができるように、様々な機能が用意されている。しかし、数値計算で使う場合、以下のようにすれば良い。この辺りについては、すぐに理解できなくても良いが、ファイル処理を多用するプログラムを作成する場合には各自勉強して欲しい。ただ、教科書にはいろいろ書かれているので、諸君がこの講義で必要なファイル処理の機能をまとめているだけである。

- 高水準ファイル処理を使え

  教科書に書いてあるとおり、高水準ファイル処理と低水準ファイル処理が、通常は前者を使う。前者はバッファーを文字や数値単位、あるいは文字単位で処理する関数が用意されており、容易にプログラムができる。それに対して、後者はバイト単位で処理するので、細かい処理ができるが扱いは面倒である。

- テキストモードを使え

  改行コードの処理により、テキストモードとバイナリーモードを使い分ける。Linux(UNIX) では C 言語と同じ改行コードを使う。したがって、テキストモードもバイナリーモードも同じ結果が得られる。Windows の場合は、テキストモードだと改行コードの変換が行われるので、注意が必要である。この辺の所は教科書を見て欲しい。

- シーケンシャルファイル処理を使え

  ファイルのデータへのアクセス方法には、シーケンシャルファイル処理とランダムファイル処理がある。前者はデータを先頭から順にアクセスし、後者は任意の場所からアクセスできる。本講義ではランダムアクセル処理を使うことはない。

- ファイル出力には、`fprintf()` 関数を使え

  `fprintf()` 関数は、標準出力 (ディスプレイ) と同じ使い方ができる。画面に出力するのと同じイメージでファイルに書き込むことができるので、容易にプログラムが書ける。実際、出力したファイルを emacs のようなエディターで見ると画面出力と同じである。

- ファイル入力には、`fscanf()` 関数を使え

  `fscanf()` 関数は、標準入力 (キーボード) と同じ使い方ができる。キーボードからデータを入力するのと同じイメージでファイルからデータを読み込むことができる。諸君は、その方法になれているので、容易にプログラムが書けるだろう。

2.3 ファイル処理の流れと関連事項

2.3.1 処理の流れ

ほとんどのプログラム言語では、ファイルの処理は図 1 のようになっている。これは人間が、データを記録している本やノートなどを見る動作と全く同じようになっている。**オープン**と**読み書き**、**クローズ**は約束事と理解して欲しい。また、同時に複数個のファイルをオープンすることも可能である。

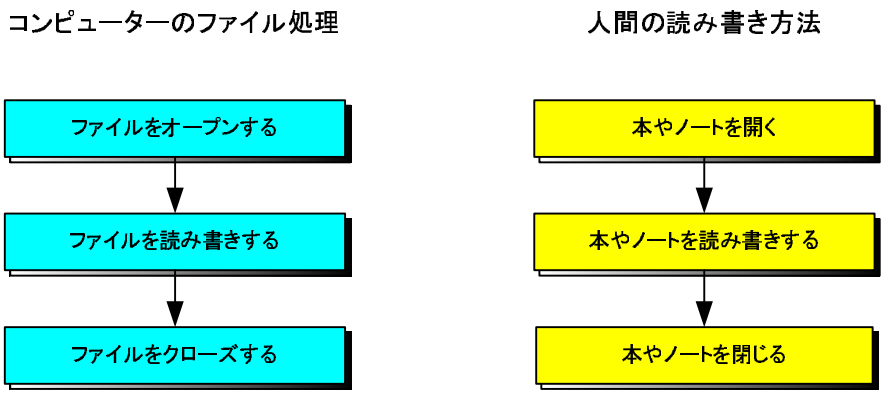

図 1: コンピューターのファイル処理と人間の読み書き方法

リスト 1 や 2 のプログラムの中で、これらの約束事の記述の例を図 2 に示す。人間の動作を考えれば、取り立ててその流れは難しくない。

- `fopen()` 関数でファイルをオープンしている。
- `fprintf()` や `fscanf()` 関数で、ファイルのデータを読み書きしている。
- `fclose()` 関数で、ファイルをクローズしている。

```
#include <stdio.h>

int main(void){
  FILE *fp;

  fp=fopen("hello.txt","w");

  fprintf(fp,"Hello World !!\n");

  fclose(fp);

  return 0;
}
```

ファイルポインターの宣言
オープン
書き出し
読み込み
クローズ

```
#include <stdio.h>

int main(void){

  FILE *fp;
  char a[32], b[32], c[32], tmp;

  fp = fopen("hello.txt", "r");

  fscanf(fp, "%s%s%s%c", a, b, c, &tmp);

  fclose(fp);

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

  return 0;
}
```

図 2: C 言語でのファイル処理の例

図 2 を見て分かるように、実際の C 言語では**オープン**と**読み書き**、**クローズ**の他に、ファイルポインターの宣言が必要である。ファイルポインターについては、次の節で述べることにする。C 言語のプログラムで

ファイル処理をする場合は、図 3 に示す手順に従えば良い。ただし、実際のプログラムではエラー処理を書かなくてはならないが、ここでは示していない。

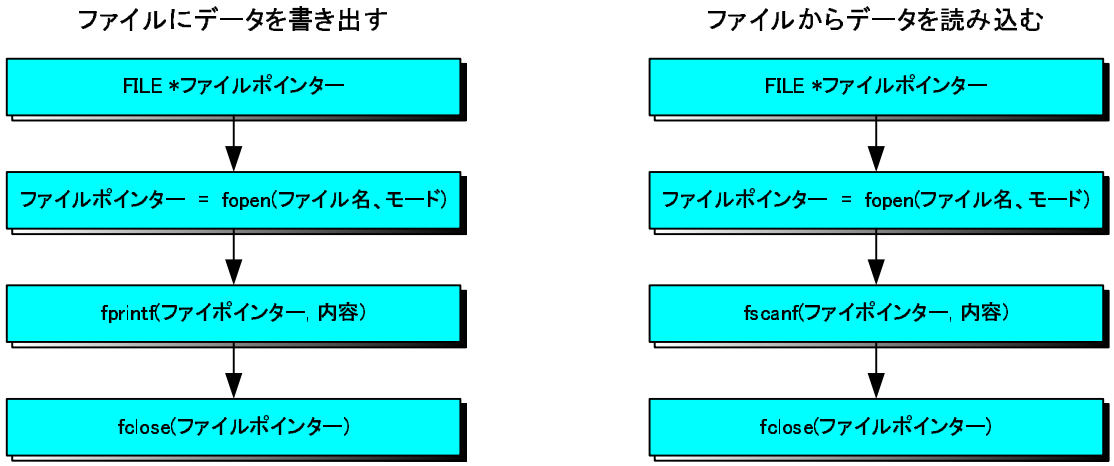

図 3: C 言語でのファイル処理の流れ

2.3.2 オープンとクローズ

C 言語では、かなり細かいファイルの処理ができる。そのために、ファイルの情報をメモリーの一部に格納する必要がある。その格納場所を示すものがファイルポインターである。それは構造体になっており、`stdio.h` というヘッダーファイルにその内容が定義されている。定義内容の例が教科書 (p.377) に書かれている。以下のようなことが記述されている。

- ファイルの読み書きをする場合の位置
- 残っている文字数
- バッファー領域の先頭位置
- ファイルの状態
- ファイル番号

すべてこのファイルポインター (`fp`) を使って、ファイル関係の処理は実施する事になる。なにせ、ファイルに関する情報が全て書かれているので、これを指定すれば、あとはコンピューターが勝手に処理してくれる。面倒くさい処理はコンピューター任せにして、プログラマーは楽をしようということである。この `FILE` 型の変数 (ポインター) を使うためには、次のように宣言する。

```
FILE *fp;
```

FILE 型の変数 (ポインター)fp を宣言したのである。ただし、fp は変数名なのでプログラマーが勝手な名前をつけて良い。

通常は、fopen() という関数の戻り値をこのポインターに代入する。このファイルをオープンする関数 fopen() の書式は、次の通りである。

FILE *fopen(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ことである。もし、オープンに失敗すると、NULL という戻り値になります。char 型のポインターと難しいことを言っているが、先のプログラムの例 (リスト 1, 2) でも分かるように、文字列をダブルクォーテーションで囲めば良いのである。例えば、hoge.txt というファイルを読み込みモードでオープンする場合

```
fp=fopen("hoge.txt", "r");
```

と書けば良い。

オープンモードについては、いろいろ用意されており、教科書の p.382 にまとめてある。細かいファイル処理をする場合は、これらのモードを巧みに使う必要があるが、本講義では、

- ファイルからデータを読み込む場合は、オープンモードの部分は"r"
- ファイルへデータを書き込む場合は、オープンモードの部分は"w"

とすればよい。バイナリーモードも UNIX(Linux) では関係ないので使わない。

ファイルをクローズする関数 fclose() の書式は、簡単で、次の通りである。

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 が返される。引数は、ファイルポインターのみである。ファイルを開いたら閉じるのが礼儀だと心得て、処理の最後に書きましょう。

2.3.3 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について説明する。難しそうですが、実は非常に簡単である。いままで、標準入力 (キーボード) と標準出力 (ディスプレイ) に使ってきた関数、printf() と scanf() とほとんど同じである。付録に示すようにキーボードやディスプレイもファイルとして取り扱われるので、同じ手法がハードディスクにも使える。

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

ファイル入力	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)

となる。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ある。非常に単純で簡単である。実際の動作もキーボードからデータを入力するのも、ファイルから読み込むのも同じイメージで取り扱える。

出力もまったく同じである。

ファイル出力　　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)

ハードディスクのファイルにデータを書き込むのは、ディスプレイにデータを出力するのと全く同じイメージである。実際、ファイル出力されたデータを見ると、ディスプレイと同じであることが分かる。

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。コンソール入出力とファイル入出力は同じ取り扱いができるのである。

2.4 ファイル出力の実際

計算結果などを大量のデータはハードディスクに保存しなくてはならない。ファイル出力のコツは、

- 表をイメージして、データをファイルに書き出す。
- 1 行に複数のデータがある場合は、"\t"を用いてタブ区切り[1]とする。
- ループ文 (for, while, do while) を用いて、fprintf() 関数を繰り返し使う。

である。

リスト 3 に三角関数の値をファイル出力するプログラムを示す。このプログラムを実行して、作成されたファイルを適当なエディター (emacs 等) で見よ。

リスト 3: 三角関数の値のファイル出力プログラム

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(void){
5    FILE *out_file;
6    double x, y1, y2, y3;
7    double pi, dphi;
8    int i, n;
9
10   pi = 4.0*atan(1.0);
11   n = 360;
12
13   dphi = 2*pi/n;
14
15   out_file = fopen("trifunc.txt","w");
16
17   for(i=0; i<=n; i++){
18     x=i*dphi-pi;
19     y1 = sin(x);
20     y2 = cos(x);
21     y3 = tan(x);
22     fprintf(out_file, "%e\t%e\t%e\t%e\n", x, y1, y2, y3);
23   }
24
25   fclose(out_file);
26
27   return 0;
28 }
```

[1]データの区切りとして、タブは良く使われる。ファイルの内容を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ある。

2.5 ファイル入力の実際

数値計算では、処理の対象となる大量のデータをハードディスクから読み込むことが多い。数値計算に限らず、現代の実験ではコンピューターのハードディスクにデータが蓄えられるのは普通である。ハードディスクに保存されたデータを読み出し、それを処理することが実験のデータ整理に必要となる。このような場合、ファイルからのデータ入力のコツは、

- データ数が多い場合、読み込んだデータは配列 (あるいは構造体) に格納する。
- ループ文を用いて、`fscanf()` 関数を繰り返し使い、データを読み込む。
- `fscanf()` 関数の戻り値が `EOF` の場合[2]、データの読み込みを止める。

である。

リスト 4 に先ほど作成したファイル内容を読み込み、ディスプレイに出力するプログラムを示す。このプログラムを実行して、ファイルの読み込みの練習をせよ。

リスト 4: ファイルの内容を読み込むプログラム

```
1  #include <stdio.h>
2
3  int main(void){
4    FILE *in_file;
5    double x[500], y1[500], y2[500], y3[500];
6    char temp;
7    int i, j;
8
9    in_file = fopen("trifunc.txt","r");
10
11   for(i=0; i<=499; i++){
12     if(
13       EOF == fscanf(in_file, "%lf%lf%lf%lf%c",&x[i], &y1[i], &y2[i], &y3[i], &temp)
14       ) break;
15   }
16
17   fclose(in_file);
18
19   for(j=0; j<i; j++){
20     printf("%f\t%f\t%f\t%f\n",x[j], y1[j], y2[j], y3[j]);
21   }
22
23   return 0;
24 }
```

3 付録

3.1 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らない。

1. ファイルポインター用の変数を FILE 型で宣言する。

[2] `EOF` は end of file の略でファイルの終わりを示す。

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、いきなりファイルの読み書きができる。コンソール入力で述べたように、通常、標準入力はキーボード、標準出力はディスプレイを示す。C 言語では (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、読み書きする。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る。そうすると、シンプルな取り扱いが可能となる。

これら、特別な 3 個のファイルについて、表 1 にまとめる。

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

`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()` と同じ動作をする。一方、`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()` と同じ動作をする。これを上手に使うと、プログラムのデバッグのときに便利である。

最後に標準エラー出力について述べる。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のことを言う。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する。`printf()` 関数を使うよりも、`fprintf()` 関数でファイルポインターとして `stderr` を指定した場合、`printf` 関数と同じ動作をする。

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

こうすると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。本当は、教科書 p.309 に書かれている `perror()` 関数を使うのがもっとも良いだろう。