ファイル

山本昌志*

2007 年 4 月 26 日

概 要

C 言語のファイル処理について学習する．ファイルの概要が分かり，それが便利であると理解する必要がある．そして，ファイルからデータを読み込み/書き込みを行うプログラムが記述できるようになることを目指す．

1 前回の復習と本日の学習内容

1.1 復習

構造体とは，関連のあるデータをひとつにまとめることができるデータ構造である．例えば，電気情報工学科 2 年生のテストの成績を表すために，次のように構造体の定義と宣言を行う．

```
typedef struct{
  char name[80];
  int math;
  int eng;
}student;

student E2[45];
```

この構造体では，名前 (name[80]) と数学の成績 (math)，英語の成績 (eng) がのメンバーである．そして，構造体の配列 E2[45] がある．

構造体のメンバーにアクセスするためには，ドット演算子をつかう．

```
strcpy(E2[12].name, "yammamoto");
E2[12].math=23;
E2[12].eng=76;

printf("%s\n",E2[12].name);
printf("%d\n",E2[12].math);
printf("%d\n",E2[12].eng);
```

*独立行政法人　秋田工業高等専門学校　電気情報工学科

1.2 本日の学習内容

本日は，ファイル処理について学習する．ハードディスクにデータを書き込んだり，ハードディスクからデータを読み込んだりする場合のプログラムの記述方法を学ぶ．

教科書 [1] の pp.326–340 が範囲である．学習範囲はここまでとするが，p.341 以降に，さらに詳しい説明がある．p.341 以降は，自分で学習せよ．

本日の学習のゴールは，以下の通りである．

- ディスプレイに文字を打ち出すのと同じイメージで，ハードディスクにデータを保存できる．そのことを理解して，データを保存するプログラムが書ける．
- キーボードからデータを読み込むのと同じイメージで，ハードディスクのデータを読み込むことができる．そのイメージを理解して，データを読み込むプログラムが書ける．

2 ファイルとは何か

2.1 ファイル

実際のプログラムでは，大量のデータを扱うことが多い．大量のデータは，ハードディスクに保存したり，ハードディスクから読み込んだりする．なぜならば，処理の結果をディスプレイに打ち出しても，書き写すことは不可能であるからである．また，大量のデータをキーボードを使って入力することもできないからである．現代社会での大量の情報は全てハードディスクに保存されている! このようなことから，ハードディスクを使ったデータ処理の方法を理解しないと，実用的なプログラムは書けない．このハードディスクとのデータの受渡しをファイル処理[1]と言う．

ハードディスクに保存しているデータの集まりの単位のことをファイル[2]と言う．これは 0 と 1 のビットの集まりで，かなり複雑なことをしてハードディスクに書き込まれている．OS—linux や Windows—がきちんと管理しているので，プログラマーは簡単に使えるようになっている．

ファイルは，図 1 にしめすような巻物と思ってほしい．この巻物は便利で，消去することができ，幅や長さはプログラマーが自由に変えることができる．ここでは，この巻物に C 言語の命令を使って，読んだり / 書いたりする方法を学習する．

[1] 正確には，ハードディスクに限らない．
[2] 正確には，ファイルシステム上のデータの集まりの単位のこと

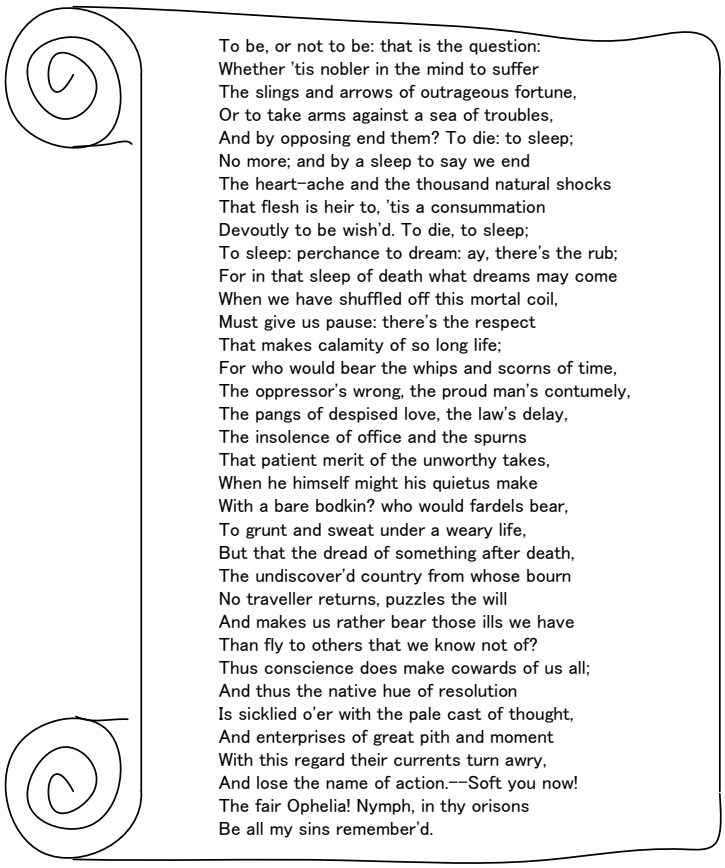

図 1: ファイルのイメージ．ファイルは巻物に似ている．巻物の本文はシェークスピアの「ハムレット」より．

2.2 ファイル処理の体験

2.2.1 ファイル出力

ごちゃごちゃと説明するよりも，実際にファイル処理のプログラムを示した方がわかりやすいだろう．いままで，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)．ディスプレイに表示する `printf()` という関数の代わりに，`fprintf()` 関数を使うのである．この関数により，ファイルに文字を書き出すことができる．ただし，この間数を使うために，前後に少し，おまけ (4,6,10 行目) がつく．

リスト 1: ファイル出力の例

```
1 #include <stdio.h>
2
3 int main(void){
4   FILE *hoge;
5
```

```
6     hoge=fopen("hello.txt","w");
7
8     fprintf(hoge,"Hello World !!\n");
9
10    fclose(hoge);
11
12    return 0;
13  }
```

実行結果

リスト 1 を実行すると，hello.txt というファイルができあがる．そのファイルの中には，以下のような文字が書かれている．

```
Hello World !!
```

このプログラムの各行の内容は，次の通りである．文法の詳細については，後で説明する．ここでは，fprintf() でファイルに文字を書くことができることを理解せねばならない．

- 4 行目 `FILE *hoge;`
  ファイルを識別するためのデータを入れる変数を準備する．
- 6 行目 `hoge=fopen("hello.txt","w");`
  hello.txt というファイルを書き込み (write) モードで開いている．そして，ファイルの情報— 正確には情報が書かれたアドレス—をポインター hoge に代入する．
- 8 行目 `fprintf(hoge,"Hello World !!\n");`
  ファイルに Hello World !!と書く．書き込むべきファイルを hoge で示している．
- 10 行目 `fclose(hoge);`
  使い終わったファイルを閉じる．

2.2.2 ファイル入力

ファイルに文字を書く方法は分かった．次に，ファイルから文字を読み込んでみよう．先ほど作成したファイル (hello.txt) の内容を読み，それを画面に出力する．リスト 2 が，ファイルを読み込んで表示するプログラムである．

リスト 2: ファイル入力の例

```
1   #include <stdio.h>
2
3   int main(void){
4     FILE *fuga;
5     char a[32], b[32], c[32];
6
7     fuga = fopen("hello.txt", "r");
8
9     fscanf(fuga, "%s%s%s", a, b, c);
10
11    fclose(fuga);
```

```
12
13    printf("%s %s %s\n", a, b, c);
14
15    return 0;
16  }
```

実行結果

このプログラムを実行すると，hello.txt というファイルから文字を読み込み，ディスプレイに，以下のように表示される．

```
Hello World !!
```

このプログラムの各行の内容は，次の通りである．ここでの文法の詳細についても後で説明する．ただ，fscanf() でファイルから文字を読み取ることができる—ことのみ理解しておけばよい．

- 4 行目 `FILE *fuga;`
  先ほど同様，ファイルを識別するためのデータを入れる変数である．
- 7 行目 `fuga = fopen("hello.txt", "r");`
  hello.txt というファイルを読み込み (read) モード開いている．戻り値であるファイルの情報は，ポインター hoge に代入している．
- 9 行目 `fscanf(fuga, "%s%s%s", a, b, c);`
  ファイルからデータ—文字列—を読み込んでいる．空白は文字列の区切りを表すので，文字列を入れる 3 つの配列を用意している．配列 a に"Hello"，配列 b に"World"，配列 c に"!!"が格納される．
- 11 行目 `fclose(fuga);`
  使い終わったファイルを閉じている．

3 ファイル処理の方法

3.1 ファイル処理の流れ

先ほどの例でファイル処理の大体の方法を理解したと思う．C 言語に限らず，ほとんどのプログラム言語のファイルの処理はどれも似かよっている．それは，図 2 のようになっている．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同じようにしている．**オープン**と**読み書き**，**クローズ**は約束事と理解する必要がある．

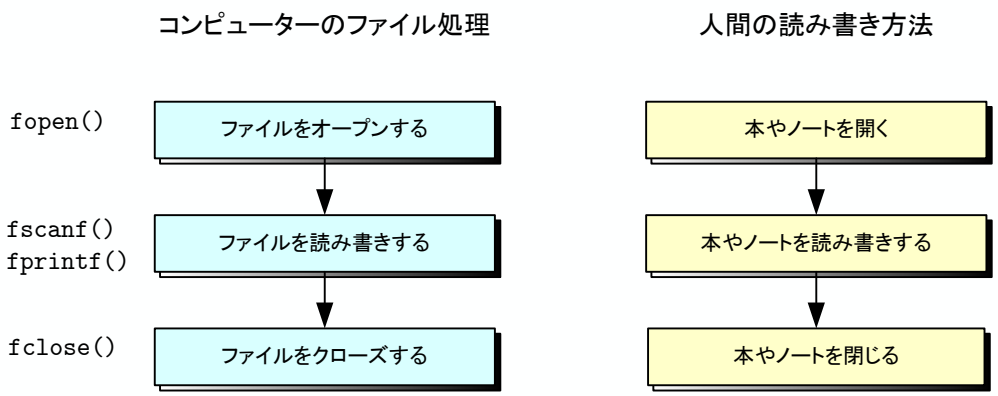

図 2: コンピューターのファイル処理と人間の読み書き方法

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 3 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．

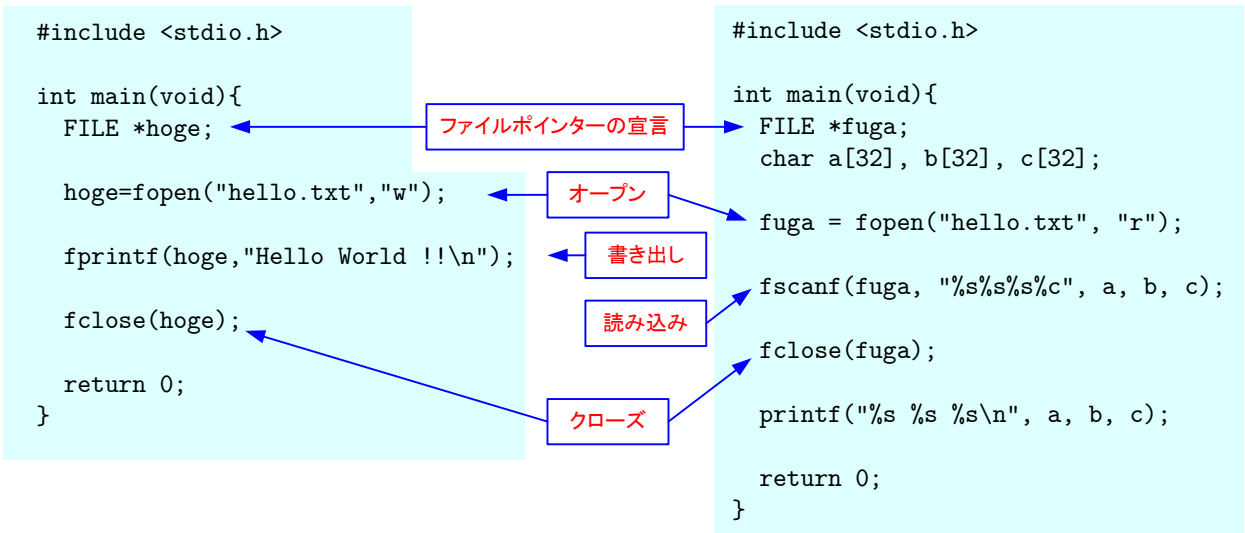

図 3: C 言語でのファイル処理の例

図 3 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 4 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．

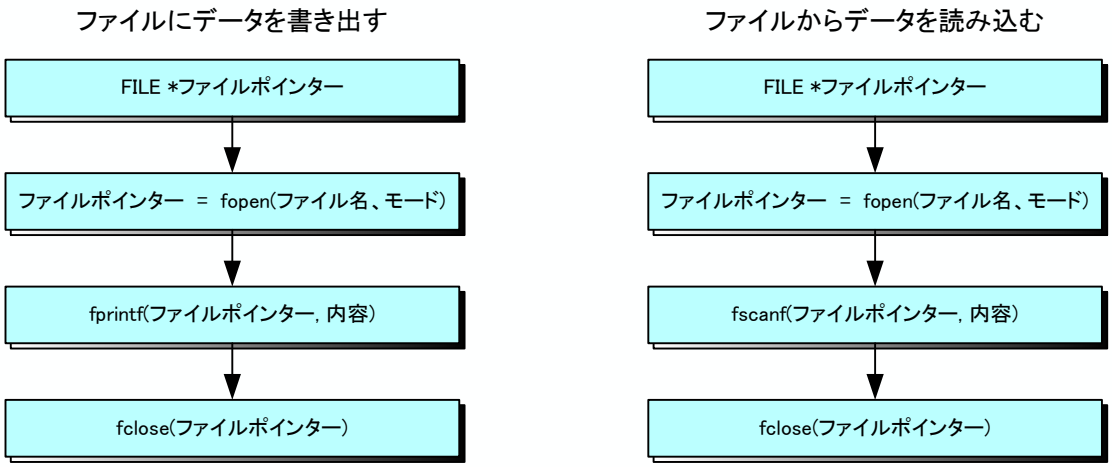

図 4: C 言語でのファイル処理の流れ

3.2 ファイルのオープンとクローズ

3.2.1 ファイルポインターの宣言

ファイル処理にはそれに関わる多くの情報が必要である．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以下のような内容が定義されている．

- 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

このファイルポインター (`fp`) を使って，全てのファイル関係の処理を行う．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 *hoge;
```

これで，FILE 型の変数 (ポインター)hoge を宣言できる．ただし，hoge は変数名なのでプログラマーが適当な名前をつける．

3.2.2 ファイルのオープン

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 という戻り値になる．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

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

と書く．

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.360 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．

3.2.3 ファイルのクローズ

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

```
fclose(foo);
```

と必ず書く．

3.3 ファイルの入出力関数

3.3.1 入出力の方法

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．難しと思うだろうが，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，入力であるが，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．戻り値の整数は入力したデータの個数である．もし，ファイルの最後あるいは読み込みに失敗すると EOF を返す．

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．戻り値の整数は出力した文字数である．書き込みに失敗すると負の値を返す．

これまでから，コンソール入出力 (キーボード入力とディスプレイ出力) とファイル入出力は同じ取り扱いができることが理解できたであろう．

3.4 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[3]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．

[3]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．

リスト 3: 三角関数の値のファイル出力プログラム

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

int main(void)
{
  FILE *out_file;           // FILE型のポインターの宣言
  double x, y1, y2, y3;
  double dphi;
  int i, n;

  n = 360;

  dphi = 2*M_PI/n;

  out_file = fopen("trifunc.txt","w");      //ファイルのオープン

  for(i=0; i<n; i++){
    x=i*dphi-M_PI;
    y1 = sin(x);
    y2 = cos(x);
    y3 = tan(x);
    fprintf(out_file, "%e\t%e\t%e\t%e\n", x, y1, y2, y3);  //書き込み
  }

  fclose(out_file);                         //ファイルのクローズ

  return 0;
}
```

3.5 ファイル入力の実際

コンピューターを使ったデータ処理では，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．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[4]，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，各列の 2 乗平均値を計算する．

リスト 4: ファイルの内容を読み込むプログラム例

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

int main(void)
{
  FILE *in_file;
  double x[500], y1[500], y2[500], y3[500];
```

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

```
 8     double sum2_x=0, sum2_y1=0, sum2_y2=0, sum2_y3=0;
 9     int i=0;
10
11
12     in_file = fopen("trifunc.txt","r");
13
14     while(fscanf(in_file, "%lf%lf%lf%lf",&x[i], &y1[i], &y2[i], &y3[i])!=EOF){
15       sum2_x += x[i]*x[i];
16       sum2_y1 += y1[i]*y1[i];
17       sum2_y2 += y2[i]*y2[i];
18       sum2_y3 += y3[i]*y3[i];
19       i++;
20     }
21
22     fclose(in_file);
23     printf("ave x^2=%f\n",sum2_x/i);
24     printf("ave y1^2=%f\n",sum2_y1/i);
25     printf("ave y2^2=%f\n",sum2_y2/i);
26     printf("ave y3^2=%f\n",sum2_y3);
27
28
29     return 0;
30   }
```

4 プログラム作成の練習

[練習 1] ファイルに以下の文を書き込むプログラムを作成せよ．そして，実行して，文が書き込まれたことを確認せよ．ただし，名前の部分は学生個人の名前を書くこと．

```
Akita National College of Technology
Department of Electrical and Computer Engineering
Masashi Yamamoto
```

[練習 2] web に載せてあるファイルをダウンロードして，そこに書かれている各列の最大値と最小値，平均値を計算せよ．

[練習 3] 以下の関数の値をファイルに書き出せ．ただし，範囲は $-1 \leq x \leq 1$，ステップ幅は 0.001 とする．

$$f(x) = x^2 \qquad g(x) = \cos(x)\sin(x) \qquad h(x) = 2^x \tag{1}$$

2^x の計算方法は，教科書の p.127 を見よ．

[練習 4] 前問で計算した関数の平均値を求めよ．

5 課題

5.1 内容

以下の課題を実施し，レポートとして提出すること．

[問 1] (復) 教科書 [1]p.326–373 を 3 回読め．レポートには「3 回読んだ」と書け．

[問 2] (復) 以下の関数の値をファイルに書き出せ．ただし，範囲は $-1 \leq x \leq 1$，ステップ幅は 0.001 とする．

$$f(x) = x^3 \qquad g(x) = \cos(x)\cos(x) \qquad h(x) = 0.5^x \tag{2}$$

[問 3] (復) 前問で計算した関数平均値を求めよ．

[問 4] (復) ここでの学習内容でわからないところがあれば，具体的に記述せよ．

[問 5] (予)C 言語によるプログラミング入門 (応用編) の第 1 章と第 2 章を 2 回読め．レポートには「2 回読んだ」と書け．

5.2 レポート 提出要領

期限　5 月 1 日 (火) AM 8:45
用紙　A4 のレポート用紙．左上をホッチキスで綴じて，提出のこと．
提出場所　山本研究室の入口のポスト
表紙　表紙を 1 枚つけて，以下の項目を分かりやすく記述すること．

授業科目名「情報処理応用」
課題名「課題　ファイル」
提出日
2E　学籍番号　氏名

内容　2 ページ以降に問いに対する答えを分かりやすく記述すること．

付録 A　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，(1)FILE 型のポインターの宣言，(2) ファイルのオープン，(3) ファイルの読み書き，(4) ファイルのクローズの処理が必要である．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．(1) と (2)，(4) の処理が不要なのである．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．同様に，`fprintf()` 関数で `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` を指定した方が後々都合が良い．

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

見た目の動作は `printf` 関数と同じ動作をする．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る．本当は，`perror()` 関数を使うのがもっとも良い．

参考文献

[1] 内田智史監修, (株) システム計画研究所編. C 言語によるプログラミング　基礎編　第 2 版. (株) オーム社, 2006.