# これまでのまとめ (後期中間テストに向けて)

山本昌志*

2004 年 11 月 26 日

## 1　制御文

順次と選択、繰り返しがプログラムの基本制御構造である。このうち、順次は、プログラムが上から下へと実行されることで実現している。従って、そのためのコマンドはない。なにもなければプログラムは上から下へと実行されるのである。残りの選択と繰り返しについて説明する。

### 1.1　選択

選択は、条件により実行される文が決まる構文である。次の 5 つの構文を学習した。

- もし、制御文が正しければ、文を実行する

書式

```
if(制御式) 文;
```

- もし、制御文が正しければ、文 1→ 文 2→ 文 3 を実行する

書式

```
if(制御式){
    文 1;
    文 2;
    文 3;
}
```

- もし、制御文が正しければ、文 1→ 文 2→ 文 3 を実行する。さもなければ (制御文が誤り)、文 4→ 文 5→ 文 6 を実行する。

*国立秋田工業高等専門学校　電気情報工学科

書式

```
if(制御式){
    文 1;
    文 2;
    文 3;
}else{
    文 4;
    文 5;
    文 6;
}
```

- もし、制御文 1 が正しければ、文 1→ 文 2 を実行する。さもなければ、制御文 2 が正しければ、文 3→ 文 4 を実行する。さもなければ、制御文 3 が正しければ、文 5→ 文 6 を実行する。さもなければ (全てが誤り)、文 7→ 文 8 を実行する。

書式

```
if(制御式 1){
    文 1;
    文 2;
}else if(制御式 2){
    文 3;
    文 4;
}else if(制御式 3){
    文 5;
    文 6;
}else{
    文 7;
    文 8;
}
```

- もし、式の値が 1 ならば文 1→ 文 2 を実行し、式の値が 2 ならば文 3→ 文 4 を実行し、式の値が 5 ならば文 5→ 文 6 を実行し、式の値がいずれでもないときは文 7→ 文 8 を実行する。

書式

```
switch(式){
  case 1:
    文 1;
    文 2;
    break;
  case 2:
    文 3;
    文 4;
    break;
  case 5:
    文 5;
    文 6;
    break;
  default:
    文 7;
    文 8;
}
```

## 1.2 繰り返し

条件が成立するまで、同じ文を何度も繰り返す構文である。

### 1.2.1 for 文

繰り返しの回数が予め分かっているときに、for 文がしばしば使われる。「初期値は○○、条件式が正しければ、ループを繰り返し、条件を再設定する。これは、条件式検査 → ループ → 条件の再設定を繰り返す。条件式が誤りになれば、そのループから抜け出す」という構文に使われる。これは、次のように、書く。

書式

```
for(初期値設定式; 継続条件式; 再設定式){
   文 1;
   文 2;
   文 3;
}
```

これは、「継続条件が正しい限り、文 1 と文 2、文 3 を実行する」となる。もし、制御式が誤り (偽) であれば、これら文は実行されず、ブロックの外側に出る。

### 1.2.2 while 文

これも、for 文同様、前判定繰り返しであるが、予め繰り返し回数が分からないときには、while 文が使われることが多い。「条件式が正しければ、ループ繰り返す。条件式が誤りになれば、そのループから抜け出す」という構文に使われる。次のように、書く。

書式

```
while(継続条件式){
   文 1;
   文 2;
   文 3;
}
```

これは、「継続条件が正しい限り、文 1 と文 2、文 3 を実行する」となる。もし、制御式が誤り (偽) であれば、これら文は実行されず、ブロックの外側に出る。

## 1.3 do while 文

これは、後判定繰り返しで、予め繰り返し回数が分からないときに使われることが多い。「ループ内を実行し、継続条件式が正しければ、さらにループを繰り返す。条件式が誤りになれば、そのループから抜け出す」という構文に使われる。次のように、書く。

書式

```
do{
   文 1;
   文 2;
   文 3;
}while(継続条件式);
```

これは、「文 1 と文 2、文 3 を実行し、継続条件が正しければ、これを繰り返す」となる。もし、制御式が誤り (偽) であれば、ブロックの外側に出る。

# 2 配列

## 2.1 データ

文字や数字等のデータを処理することがコンピューターの仕事と考えることができる。処理すべきデータは、全て、コンピューターのメモリー[1]に記憶しなくてはならない。それにアクセスするためには、プログ

[1] 主記憶 (メインメモリー) と呼ばれ、直接 CPU がアクセスする。プログラムも実行時ここに格納される。

ラマーはその記憶場所に名前を付けなくてはならない。加えて、C 言語の場合[2]、効率よくデータを扱う[3]ためにデータの型も指定する必要がある。

## 2.2 単純型のデータ構造

最初に学習した単純型のデータ構造の場合、

```
int a;
double x;
```

のように宣言すると、

- a と名付けられた整数型のデータ領域が一つ用意される。
- x と名付けられた倍精度実数型のデータ領域が一つ用意される。

となる。このデータ構造では、変数名、たとえば a や w を指定することで、その領域のデータを入出力できる。

## 2.3 配列

単純型のデータ構造の場合、一度に確保できるメモリーの領域は 1 個なので、大量のデータを扱うのは不向きである。そこで、大量のデータを扱うために、配列というデータ構造が考えられた。

これは、同じ型のデータを任意の個数宣言し、配列名と自然数[4]でアクセスすることができ、便利である。配列を使うためには、

```
int b[10000];
double y[10000];
```

のように宣言をする。こうすると、

- 配列名 b の整数型のデータ領域が 10000 個用意される。用意されるデータ領域は、b[0]〜b[9999] である。
- 配列名 y の倍精度実数型のデータ領域が 10000 個用意される。用意されるデータ領域は、y[0]〜y[9999] である。。

となる。このデータ構造では、配列名と添え字 (インデックス)、たとえば b[1234] や y[45] を指定することで、その領域から値を入出力できる。

配列型のデータを取り扱う場合、繰り返し文とともに使われることが多い。次の例のようにである。

[2] C 言語に限らず多くのプログラミング言語で、変数の型の宣言は必要である
[3] コンパイラーやハードウェアーの都合である。
[4] ここでは、0 も自然数に含める。

```
for(i=0; i<=9999;i++){
  y[i]=3.1415*b[i]
}
```

添え字が 1 つのものを一次元配列と言い、それ以上のものを多次元配列と言う。C 言語では多次元配列を使う場合、

```
int hoge_1[100], hoge_2[100][100], hoge_3[100][100][100];
double huga[10], huge[10][10], hugo[10][10][10];
```

のように宣言を行う。これらも、配列名と複数の添え字で、そこにあるデータにアクセスする事ができる。3 次元以上ももちろん可能である。

# 3 ファイル入出力

## 3.1 取り扱い手順

コンピューターで大量のデータを操作する場合、ハードディスク上のファイルの取り扱いが必須である。ハードディスク上のファイルを取り扱うプログラムは簡単で、

1. FILE 型の変数、ファイルポインターの用意
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

を記述すれば良い。

## 3.2 ファイル出力

簡単な例として、次の

- ファイル名は、"test_out.txt"とする。
- そこに、2004 と整数を書き込む。

ようなハードディスクにデータを書き込むプログラムを示す。このプログラムは、以下にように書く。

```
#include <stdio.h>

int main(void)
{
  FILE *fp_write;
```

```
  fp_write = fopen("test_out.txt","w");

  fprintf(fp_write,"%d", 2004);

  fclose(fp_write);

  return 0;
}
```

簡単である。ファイルポインターの宣言とオープンとクローズはおまじないと思えば良い。実際に、ファイルにデータを書き出す部分は、fprintf 関数を使う。これは、ディスプレイに出力する printf 関数とほとんど同じである。

## 3.3 ファイル入力

次の例は、ハードディスクからデータを読み込むプログラムである。

- 読み込むファイルは、3.2 節で作った"test_out.txt" とする。
- ファイルに書かれている整数を書き込み、変数 a に格納する。

このプログラムは、以下にように書く。

```
#include <stdio.h>

int main(void)
{
  FILE *fp_read;
  int a;

  fp_read = fopen("test_out.txt","r");

  fscanf(fp_read,"%d", &a);

  fclose(fp_read);

  return 0;
}
```

難しいことは何も無い。おまじないの部分は、データ出力と同じである。ファイルからデータを読み出す部分は、fscanf 関数を使う。これは、キーボードから読み込む scanf 関数とほとんど同じである。

## 3.4 エラー処理付きファイルオープン

実用的なプログラムでは、エラー処理は必須である。ファイル操作のプログラムの場合、ファイルが無い等の理由でオープンできない場合がある。このようなときには、エラー処理として

```
if((fp_read = fopen("test_out.txt","r"))==NULL){
  printf("ファイルが開けません\n");
  return 1;
}
```

と書く。この例は、ファイル読み込みの場合であるが、書き込みの場合も同じようにエラー処理を書く。これはファイル処理をする場合のオープンのパターンと憶えておいて欲しい。

# 4 プログラム例

## 4.1 繰り返し文

以下のプログラムが作成できること。

- `for` 文を用いて、1〜10000 までの和を計算するプログラム
- `while` 文を用いて、1〜10000 までの和を計算するプログラム
- `do while` 文を用いて、1〜10000 までの和を計算するプログラム

## 4.2 アクセスカウンター

### 4.2.1 問題

教科書の P.186 の Lesson 5-2 の問題である。

> ホームページを閲覧しているときに、アクセスカウンターと呼ばれる「そのページが何回閲覧されたか」を表示する仕組みがある。これは、ホームページにアクセスしたときにあるプログラムが動くようになっていて、そのプログラムが何回実行されたかを調べている。すなわち、実行回数をファイルに保存しておき、プログラムを実行するときにはそのファイルを読み込んで実行回数に 1 を加えてまた保存するということを行っているわけだ。
>
> このようなときに利用できるアクセスカウンタープログラムの基本として、プログラムの実行回数をカウントし、「○回目の実行です」と表示するプログラムを作成しなさい。

ただし、

- アクセス数を記述したファイルは、`/tmp/1e/up.log` にある。

とする。

### 4.2.2 ソースプログラム

次のようなソースプログラムが書ける。これをよく理解すること。

```
#include <stdio.h>

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

  /*--------------- ファイルのオープン（読み込みモード） ----------------------*/

  if((fp=fopen("/tmp/1e/up.log","r"))==NULL){
    printf("can not open the file\n");
    return 1;
  }

  fscanf(fp, "%d", &access);    /*-- ファイルからデータを読み込む --*/

  fclose(fp);                   /* -- ファイルのクローズ --*/



  /*--------------- 実行回数の表示 ---------------------------------------*/

  printf("%d 回目の実行です\n", access);


  /*--------------- ファイルのオープン（書き込みモード） ----------------------*/

  if((fp=fopen("/tmp/1e/up.log","w"))==NULL){
    printf("can not open the file\n");
    return 1;
  }

  fprintf(fp, "%d",access+1);     /*-- ファイルへデータを書き込む --*/

  fclose(fp);                     /*-- ファイルのクローズ --*/

  return 0;

}
```

## 4.3 温度のデータ処理

### 4.3.1 問題

問題として与えられたのは、以下の条件のプログラムの作成である。

- 11 月の毎日の 1 時間毎の気温の表図 1 のようなデータファイル (/tmp/1e/temperature.txt) がある。
  - 各行には、その日の 1 時間毎の 24 個のデータがある。0 時〜23 時までである。

  - 行数は 30 行で、11 月 1 日から 11 月 30 日を表している。
  - ファイルには、温度のみが書かれている。時刻や日にちは書かれていない。
- 日毎の最高気温と最低気温、平均気温をディスプレイに書き出す。
- 11 月の最高気温と最低気温、平均気温をディスプレイに書き出す。

表 1: 11 月の気温

| | 0 時 | 1 時 | 2 時 | 3 時 | ··· | 23 時 |
|---|---|---|---|---|---|---|
| 1 日 | 8.3 | 7.9 | 7.5 | 7.2 | ··· | 9.8 |
| 2 日 | 9.3 | 9.2 | 9.1 | 9.0 | ··· | 6.3 |
| 3 日 | 6.2 | 5.8 | 5.3 | 4.9 | ··· | 12.0 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋱ | ⋮ |
| 30 日 | 4.3 | 3.9 | 3.3 | 2.8 | ··· | 3.8 |

#### 4.3.2 ソースプログラム

次のようなソースプログラムが書ける。これをよく理解すること。

```
#include <stdio.h>

int main(void){
  FILE *fp;
  double temp[31][24];
  double max_day[31], min_day[31], av_day[31];
  double max_nov, min_nov, av_nov;
  double sum_day, sum_nov;
  int dates, hours;
  int i, j;

  /* --------------- 日数と時間の設定 --------------------------*/

  dates = 30;
  hours = 24;

  /* --------------- ファイルのオープン ------------------------*/

  if((fp=fopen("/tmp/1e/temperature.txt","r"))==NULL){
    printf("can not open the file\n");
    return 1;
  }


  /* --------------- ファイルからデータを読み込む ---------------*/
```

```
for(i=1; i<=dates; i++){
  for(j=0; j<=hours-1; j++){
  fscanf(fp, "%lf", &temp[i][j]);
  }
}


/* --------------- ファイルのクローズ ----------------------*/

fclose(fp);


/* --------------- 最大と最小、平均の計算 --------------------*/

max_nov = -9999;                    /* 11 月の仮の最高気温 */
min_nov = 9999;                     /* 11 月の仮の最低気温 */
sum_nov = 0.0;                      /* 11 月の気温の合計の初期値 */

for(i=1; i<=dates; i++){            /* i は、日にちを表す */

  max_day[i] = -9999;               /* i 日の仮の最高気温 */
  min_day[i] = 9999;                /* i 日の仮の最低気温 */
  sum_day = 0.0;                    /* i 日の気温の合計の初期値 */

  for(j=0; j<=hours-1; j++){        /* j は時刻を表す      */
    sum_day += temp[i][j];          /* 時刻毎の気温の合計 */

    if(temp[i][j] > max_day[i]){    /* i 日の最大気温の探索  */
      max_day[i] = temp[i][j];
    }

    if(temp[i][j] < min_day[i]){    /* i 日の最低気温の探索  */
      min_day[i] = temp[i][j];
    }
  }

  av_day[i]=sum_day/hours;          /* i 日の平均気温の計算 */

  sum_nov += av_day[i];             /* 11 月 1 日までの平均気温の和 */

  if(max_day[i] > max_nov){         /* 11 月の最大気温の探索 */
    max_nov = max_day[i];
  }

  if(min_day[i] < min_nov){         /* 11 月の最低気温の探索 */
    min_nov = min_day[i];
  }
```

```
    }

    av_nov = sum_nov/dates;           /* 11 月の平均気温の計算 */


    /* --------------- 結果の表示 ----------- --------------------*/

    printf("\n\nTemperature  November/2003 at Akita\n");
    printf("-----------------------------------\n");
    printf(" day     max     min    average \n");
    printf("===================================\n");

    for(i=1; i<=dates; i++){
      printf("%3d     %5.1lf   %5.1lf   %5.1lf\n",
      i, max_day[i], min_day[i], av_day[i]);
    }

    printf("-----------------------------------\n\n");

    printf("max(Nov.)     = %5.1lf\n", max_nov);
    printf("min(Nov.)     = %5.1lf\n", min_nov);
    printf("average(Nov.) = %5.1lf\n\n", av_nov);


    return 0;

  }
```

#### 4.3.3 実行結果

実行結果、以下のようになる。

```
Temperature  November/2003 at Akita
-----------------------------------
 day     max     min    average
===================================
  1      20.3     6.0    12.1
  2      20.4     9.5    15.2
  3      22.8    13.9    17.8
  4      14.9     6.4    11.5
  5      18.4     4.3    10.6
  6      17.9    10.5    13.3
  :         :       :       :
  :         :       :       :
 25      10.6     7.6     9.2
 26       7.0     0.6     4.3
 27       7.3    -0.3     2.8
 28      12.2     0.1     6.0
 29      15.7     7.9    12.2
 30      15.1    11.2    13.4
-----------------------------------
```

```
max(Nov.)     =  22.8
min(Nov.)     =  -0.5
average(Nov.) =   9.2
```