C 言語の学習
関数

山本昌志*

2004 年 5 月 26 日

1　本日の学習内容

本日の内容は、教科書の 11 章である。以下のことをよく理解して欲しい。

11 章　関数

- 関数を使う理由が分かる。
- 関数の書き方が分かる。
- データの受け渡しが分かる。

2　関数 (11 章)

2.1　関数とは何か

2.1.1　関数の役割

1 年生の時に学習した FORTRAN を覚えている人は、サブルーチンを思い出して欲しい。それが、C 言語の関数に相当する。いかなるプログラミング言語でも、このサブルーチンに相当する機能がある。それだけ便利な機能であるということである。

C 言語のプログラムは、プリプロセッサー (15 章) と宣言文と、実行文からなる。これらのうちプログラム実行中に CPU が動作するのは実行文の処理だけである。後に学習するが、プリプロセッサーはコンパイル時 (gcc コマンド使用時) の動作に使われる。宣言文である変数宣言などは、プログラムがメモリーにロードされる際にそれを予約するのに使われるだけである。メモリーにロードされたプログラムが実際に動作するのであるが、その動作内容は実行文に従う。コンピューターは、プログラム中の実行文の順序に従い、自動的にメモリーの内容を処理しているだけである。

この実行文は、必ず関数のブロック中に書かれなくてはならない。今まで、学習してきたプログラムでは、`main` 関数のみがあり、その中に実行文が書かれていたはずである。`main` 関数は特別で、C 言語のプログラムには、必ず 1 個必要で、そこから実行されることになっている。

*独立行政法人　秋田工業高等専門学校　電気工学科

実際の C 言語のプログラムは、main 関数のみからできていることは希で、リスト 1 のように複数の関数からできている。

リスト 1: 処理を関数でまとめたプログラム

```
1  #include <stdio.h>
2
3  double max(double a, double b);
4
5  /*======================================================================*/
6  /*      メイン関数                                                      */
7  /*======================================================================*/
8  int main(){
9    double x, y, z;
10
11   x=2.5;
12   y=3.1415;
13
14   z=max(x, y);
15
16   printf("大きい方は、%eです。\n", z);
17
18   return 0;
19 }
20
21 /*======================================================================*/
22 /*      大きい方を探す関数                                              */
23 /*======================================================================*/
24 double max(double a, double b){
25   double big;
26
27   if(a<b){
28     big = b;
29   }else{
30     big = a;
31   }
32
33   return big;
34
35 }
```

関数は、長いプログラムを効率よく記述するために必要である。そのために、この関数には 2 つの役割がある。一つは、同じような処理を一つにまとめることである。実際のプログラムの動作は、同じ処理、あるいは似たような処理が非常に多い。いちいちそれを書くとプログラムが長くなり、プログラマーは大変である。そこで、一つにまとめ、必要なときに呼び出すのである。

もう一つ、長いプログラムの問題は、処理が分かりにくい点である。例えば、windows2000 だとソースプログラムは大体 4000 万行だと言われている。この場合、それぞれの実行文の役割など分からない。コンピューターは大量のトランジスターからできているが、それぞれの役割が分からないのと同じである。このように大量の部品 (実行文) から構成されるコンピューター (プログラム) の動作を考える際に重要なことは、モジュール[1]に分解することである。そうすると、動作の内容が分かるようになる。長いプログラムを作る場合も同じで、機能単位 (モジュール) に分け、分かりやすくすることが重要である。C 言語では関数を使い、機能単位にプログラムを分割する。

まとめると、関数の役割は

[1]module:単位。機器の場合、ある機能を果たす部品単位を表す

- 何度も同じことを書くのは面倒なので、同じような処理を一つにまとめる。
- プログラムの内容を分かりやすくするために、処理を機能単位に分割する。

である。特に、2 番目が重要で、「プログラムのソースは分かりやすくなくてはならない」ということを、肝に銘じておかなくてはならない。

[**練習 1**] リスト 1 をプリプロセッサーの文と宣言文、実行文に分けよ。

[**練習 2**] リスト 1 のプログラムの実行順序を考えよ。

2.1.2 同じ処理をまとめる

関数の役割の一つの「同じ処理をまとめる」ことの例を示す。そのために、大きさ 10 の配列 a[] と大きさ 100 の配列 b[] に整数の乱数を格納し、その最値を求めるプログラムを考える。リスト 2 のようなプログラムである。

リスト 2: 関数を使わない最大値検索プログラム

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int main(void){
6    int a[10], b[100], max_a, max_b, i;
7
8    srand((unsigned) time(NULL));           /* 起動毎に異なる乱数発生のため */
9
10   for(i=0; i<10;   i++) a[i]=rand();      /* 配列 a[] の値設定 */
11   for(i=0; i<100;  i++) b[i]=rand();      /* 配列 b[] の値設定 */
12
13   /* ---- 配列 a[] の最大値検索 ----- */
14   max_a=a[0];
15   for(i=1; i<10; i++){
16     if(max_a<a[i]) max_a=a[i];
17   }
18
19   /* ---- 配列 b[] の最大値検索 ----- */
20   max_b=b[0];
21   for(i=1; i<100; i++){
22     if(max_b<b[i]) max_b=b[i];
23   }
24
25   printf("max a=%d\n",max_a);             /* 最大値印刷 */
26   printf("max b=%d\n",max_b);
27
28   return 0;
29 }
```

リスト 2 をよく見ると、最大値を求める部分はほとんど同じである。そこで、それを一つの関数にまとめることを考える。リスト 3 のようにすれば良い。これで、プログラムがすっきりした。こうすると、最大値を求めるアルゴリズムを変えたい場合でもプログラムの変更が容易である。

リスト 3: 関数を使用した最大値検索プログラム

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

int find_max(int n, int ix[]);              /* プロトタイプ宣言 */

/*=========================================================================*/
/*    main 関数                                                            */
/*=========================================================================*/
int main(void){
  int a[10], b[100], max_a, max_b, i;

  srand((unsigned) time(NULL));            /* 起動毎に異なる乱数発生のため */

  for(i=0; i<10;   i++) a[i]=rand();       /* 配列 a[] の値設定 */
  for(i=0; i<100;  i++) b[i]=rand();       /* 配列 b[] の値設定 */

  max_a=find_max(10, a);
  max_b=find_max(100,b);

  printf("max a=%d\n",max_a);              /* 最大値印刷 */
  printf("max b=%d\n",max_b);

  return 0;
}

/*=========================================================================*/
/*    最大値探索の関数                                                     */
/*=========================================================================*/
int find_max(int n, int ix[]){
  int i, max;

  /* ---- 配列 a[] の最大値検索 ----- */
  max=ix[0];
  for(i=1; i<n; i++){
    if(max<ix[i]) max=ix[i];
  }

  return max;
}
```

2.1.3 処理をまとめてプログラムを分かりやすく

関数のもう一つの機能「処理をまとめてプログラムを分かりやすく」の例である。リスト 2 のメイン関数の部分の処理を分かり易くするために、関数を用いた例をリスト 4 に示す。

リスト 4: 処理を関数でまとめたプログラム

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

/* ----  プロトタイプ宣言 -----  */
void make_data(int nx, int ix[], int ny, int iy[]);
int find_max(int n, int ix[]);
void print_results(int a, int b);

```

```
10 /*=========================================================================*/
11 /*    main 関数                                                            */
12 /*=========================================================================*/
13 int main(void){
14   int a[10], b[100], max_a, max_b;
15
16
17   make_data(10, a, 100, b);                /* 乱数データ作成 */
18
19   max_a=find_max(10, a);                   /* 最大値検索 */
20   max_b=find_max(100,b);
21
22   print_results(max_a, max_b);             /* 最大値印刷 */
23
24   return 0;
25 }
26
27
28 /*=========================================================================*/
29 /*    データ作成                                                           */
30 /*=========================================================================*/
31 void make_data(int nx, int ix[], int ny, int iy[]){
32   int i;
33
34   srand((unsigned) time(NULL));           /* 起動毎に異なる乱数発生のため */
35
36   for(i=0; i<nx;    i++) ix[i]=rand();
37   for(i=0; i<ny;    i++) iy[i]=rand();
38
39 }
40
41
42 /*=========================================================================*/
43 /*    最大値探索の関数                                                     */
44 /*=========================================================================*/
45 int find_max(int n, int ix[]){
46   int i, max;
47
48   /* ---- 配列 a[] の最大値検索 ----- */
49   max=ix[0];
50   for(i=1; i<n; i++){
51     if(max<ix[i]) max=ix[i];
52   }
53
54   return max;
55 }
56
57
58 /*=========================================================================*/
59 /*    結果の印刷                                                           */
60 /*=========================================================================*/
61 void print_results(int a, int b){
62
63   printf("max a=%d\n",a);
64   printf("max b=%d\n",b);
65
66 }
```

2.2 関数の作り方と使い方

関数を使うためには、以下の 3 つのことが必要である。図 1 を見ながら、以下を理解せよ。

- プロトタイプ宣言
  - 戻り値と関数名、それから引数を書く。
  - 関数が正しく使われているか、コンパイラーがチェックするために用意されている。
- 関数呼び出し
  - 関数を呼び出すためには、引数を伴って関数名をコールするだけである。
  - どこからでも、何回もコールすることができる。
- 関数定義
  - メイン関数と同様、処理内容を書けば良い。
  - 呼び出し元からのデータは引数で渡される。
  - return 文で呼び出し元へ、データを送る。その書き方は、次の 2 つが用意されている。変数のみも式と見なせる。
    - 括弧無しの方法　`return 式;`
    - 括弧付きの方法　`return(式);`

2.3 データを渡す方法 (p.214)

関数を使って処理を行う場合、呼び出す側の関数とと呼び出される側の関数とでデータの受け渡しが必要である。呼び出す側は値 (あるいは変数) を用意し、呼び出される側は変数を用意する。呼び出す側の値を呼び出される側の変数にコピーすることで、データの受け渡しが行われる。呼び出す側の関数の値 (変数) を実引数、呼び出される側の関数の変数を仮引数と言う。図 1 に示しているので、実引数と仮引数の違いを理解せよ。

関数へのデータの渡し方に、2 つの方法がある (通常は値渡し)。

値渡し　呼び出す側と呼ばれる側の関数が各々変数を用意する。仮引数側の変数の値は、実引数の側の変数の値のコピーとなる。呼ばれた関数が変数の値を変えても、呼び出した側の変数値には、影響がない。

アドレス渡し　呼び出す側の実引数は、アドレスです。呼ばれる側は、ポ インターを用意して、実引数のアドレスを受け取ります。呼ばれた関 数が処理をすると、呼び出した側の実引数の変数にも影響があります。

それでは、データの受け渡しについて、教科書を見ながら、練習せよ。

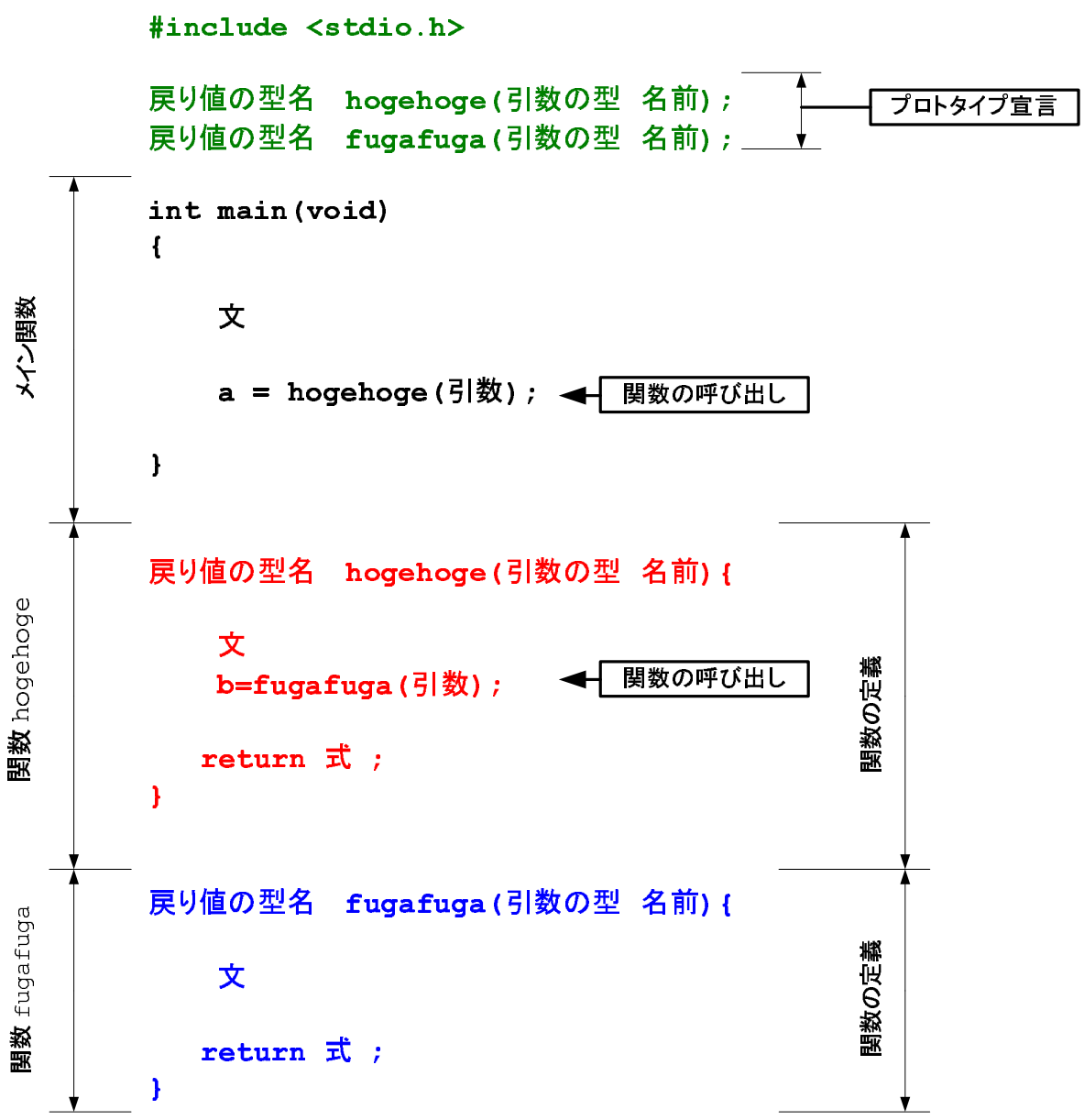

図 1: 関数を使ったプログラムの基本的な書き方

2.3.1 値による呼び出し (p.214)

以下の練習問題を実施せよ。

[練習 1] main 関数から、角度 [度] の値を関数に渡して、プログラマー作成の関数で、sin と cos と tan を計算して、印刷する。このプログラムを作成せよ。ただし、`main` 関数で繰り返し文を使い、0〜360[度] まで、1 度間隔でデータを送ること。ヒントは以下の通り。

- `#include <math.h>`を書く。三角関数のような数学関数を使う場合、`math.h` というヘッダーファイルが必要である。
- 三角関数は、`sin(a)`,`cos(a)`,`tan(a)` と書けば計算できる。単位は、[ラジアン] である。
- コンパイルには、`-lm` オプションが必要である。このオプションは数学関数を使うときに必要である。実際には次のようにする。C 言語のソースファイルが `hogehoge.c` で、機械語の実行ファイルが `fugafuga` である。

  `gcc -lm -o fugafuga hogehoge.c`

[練習 2] リスト 5 の変数 `i,j` の値が入れ替わらない理由を考えよ。

リスト 5: 値渡しを用いたため、値が交換されない例

```
1  #include <stdio.h>
2
3  void swap(int i, int j);
4
5  int main(){
6    int a=2, b=3;
7
8    swap(a, b);
9
10   printf("a=%d b=%d\n", a, b);
11
12   return 0;
13 }
14
15
16 void swap(int i, int j){
17   int temp;
18
19   temp = i;
20   i=j;
21   j=temp;
22 }
```

2.3.2 アドレスによる呼び出し (p.216)

以下の練習問題を実施せよ。

[練習 1] リスト 6 の変数 `i,j` の値が入れ替わる理由を考えよ。

リスト 6: アドレス渡しを用いたため、値が交換さる例

```c
#include <stdio.h>

void swap(int *i, int *j);

int main(){
  int a=2, b=3;

  swap(&a, &b);

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

  return 0;
}


void swap(int *i, int *j){
  int temp;

  temp = *i;
  *i=*j;
  *j=temp;
}
```

2.3.3 一次元配列を渡す (p.218)

以下の練習問題を実施せよ。

[**練習 1**] 要素数が 10000 個の一次元配列に、整数の乱数を格納して、その最大値を求めるプログラムを作成せよ。

- 最大値を求めるルーチンは、独立の関数とせよ。
- その関数では、最大値の出力も行うこと。

2.3.4 多次元配列を渡す (p.220)

以下の練習問題を実施せよ。

[**練習 1**] 要素数が 1000×1000 の二次元配列に、整数の乱数を格納して、その最大値を求めるプログラムを作成せよ。

- 最大値を求めるルーチンは、独立の関数とせよ。
- その関数では、最大値の出力も行うこと。

2.4 戻り値を返す方法 (p.222)

2.4.1 1 個の戻り値を返す (p.222)

以下の練習問題を実施せよ。

[**練習 1**] 教科書の例を参考にして、

$$f(x) = x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040}$$

を計算する関数を作成せよ。そして、メインルーチンで、$x = 0$〜$x = 3.14$ まで、180 等分した値を求めよ。この値は何か?。注意事項は以下の通り。

- n 乗は、pow と言う関数をつかう。使い方は、教科書の P.437 を見よ。
- pow を使うためには、`#include <math.h>`が必要である。さらにコンパイルするときには、-lm オプションが必要である。

2.4.2 複数の戻り値 (p.223)

以下の練習問題を実施せよ。

[**練習 1**] $\sin 2\theta$ と $2\sin\theta\cos\theta$ の値を 0〜360 で 1 [度] 間隔で、を計算するプログラムを作成せよ。

- ただし、これらの三角関数の値は、一つの関数で計算すること。
- プロトタイプの宣言は、以下のようにすること。

```
void keisan(double theta, double *s1, *s2);
```

2.5 配列の受け渡し

配列の受け渡しは、変数の場合と異なる。1 変数の場合、実引数のコピーが仮引数に渡されることは、以前に述べた通りである (値渡し)。しかし、驚いたことに、配列の場合、実引数のアドレスが、仮引数に渡される。FORTRAN と同じである。なぜ、このような不統一が生じるかというと、

- 配列を使う場合、そのサイズが大きい場合が多い。大きな配列をコピーして仮引数に渡すと、非常にコストがかかる。コストというのは、CPU 時間やメモリーのこと。したがって、配列のコピーは不経済なので、避けなくてはならない。

ということが理由のようである。仕様でそのようになっているので仕方ないが、不統一の感はぬぐない。この辺が、C 言語は難しいと思う人が出てくる原因になっている可能性がある。

この辺のことをリスト 7 でみる。関数 transpose で行列を転置[2]している。main 関数の transpose(a) で、関数 transpose に配列 a[5][5] の先頭アドレスを渡している。配列のところで学習したように、配列名は、その配列の先頭アドレスを表す。

先頭アドレスは、transpose 関数では、

```
transpose(int x[][5])
```

[2]行と列を入れ替えること。

の形で受け取る。アドレスが渡されたので、ポインターで受け取ることも出来るが、そうすると、後々の操作が大変になるので通常は避ける。配列を受け取る場合は、配列として受け取る方が断然良い。

ここで、受け取る配列の大きさが、x[][5] とすべて記述されていない。配列のところで、学習したように、サイズは、指定の配列をアクセスするためのアドレスのオフセット計算に用いる。したがって、オフセット計算に不用な一番左のサイズは書かなくても良い。もちろん書いても良く、twx[5][5] でも問題はないが、間違える確率が増えるので通常は書かない。。

一方、配列定義のときは、コンパイラーはメモリーを確保するために、配列全体サイズが必要である。したがって、x[5][5] のようにすべて記述しなくてはならない。このような理由から、もっと多次元の配列、例えば、x[100][100][5][5] のような配列を hogehoge 関数側で受け取る場合、hogehoge(int x[][100][5][5]) と書く。左端のみ省略可である。

リスト 7: 配列の受け渡し方法

```
1  #include <stdio.h>
2  void transpose(int x[][5]);
3
4  /* ----------------- main ----------------------*/
5  int main(void){
6    int i;
7    int a[5][5]
8      ={{11, 12, 13, 14, 15},
9        {21, 22, 23, 24, 25},
10       {31, 32, 33, 34, 35},
11       {41, 42, 43, 44, 45},
12       {51, 52, 53, 54, 55}};
13
14   for(i=0; i<=4; i++){
15     printf("%d %d %d %d %d \n",
16            a[i][0], a[i][1], a[i][2], a[i][3], a[i][4]);
17   }
18
19   transpose(a);
20   printf("\n");
21
22   for(i=0; i<=4; i++){
23     printf("%d %d %d %d %d \n",
24            a[i][0], a[i][1], a[i][2], a[i][3], a[i][4]);
25   }
26
27   return 0;
28
29 }
30
31 /* -------------- transpose function -------------*/
32 void transpose(int x[][5])
33 {
34   int i, j, temp;
35
36   for(i=0; i<=3; i++){
37     for(j=i+1; j<=4; j++){
38       temp = x[i][j];
39       x[i][j] = x[j][i];
40       x[j][i] = temp;
41     }
42   }
43 }
```

結果

```
11 12 13 14 15
21 22 23 24 25
31 32 33 34 35
41 42 43 44 45
51 52 53 54 55

11 21 31 41 51
12 22 32 42 52
13 23 33 43 53
14 24 34 44 54
15 25 35 45 55
```

2.6 グローバル変数によるデータの受け渡し (p.228)

グローバル変数を使うと関数の独立性が失われ、再利用のする場合、支障をきたす。独立性の高い関数はコピーすると、そのまま他のプログラムでも使える。グローバル変数を使うと、関数のみならず、グローバル変数もコピーする必要が生じる。さらに、グローバル変数は全ての関数で使えるため、名前の衝突を考えなくてはならない。大きなプログラムになると、これは大変な問題を生じる。非常に分かりにくいバグの原因となるので、気を付けなくてはならい。

この講義で諸君が作る程度の短いプログラムならば、グローバル変数を使っても良いだろう。むしろポインターが分からなくて悩むよりは、グローバル変数を使った方がプログラムを楽しめて良いだろう。

[練習 1] リスト 6 のプログラムをポインターを使わないでグローバル変数を使ったプログラムに書き換えよ。

2.7 `main` 関数への引数渡し (p.228)

教科書に書かれているこのテクニックは、よく使われる。しかし、本講義ではあまり使わないので説明しない。興味のある者は、自分で調べよ。