関数

山本昌志*

2005 年 1 月 21 日

1　本日の学習内容

本日は、サブルーチンがなぜ必要か学習して、それを C 言語で実装する方法を学習する。これを学ぶことにより、かなり長いプログラムが書けるようになる。

1.1　良いソースコード とは

良いプログラムは、時代とともに変わってきている。それは、コンピューターのハードウェアーの進歩に応じて、求められるプログラムの書き方が変わるのである。プログラマーはハードウェアーの能力の応じたソースコードを書く必要がある。

昔は、CPU の能力は貧弱で、使えるメモリーも少なかった。当時は、速度が速く、メモリーを余り使わないプログラムが良いとされた。そのためには、かなりアクロバティックなプログラムが書かれたようである。高級言語に機械語を併用したり、メモリーに関しては様々な節約テクニックを駆使したようである。このようなプログラムのソースは必然的に内容が分かりにくくなる。

現在では、パソコンですら、ありあまる CPU 能力と巨大なメモリー空間が利用できるようなっている。しかしながら、Graphical User Interface(GUI) が当たり前になり、プログラムそのものも巨大なものとなってきている。プログラムの巨大化に合わせて、プログラマー能力が進歩すれば良いのであるが、それは不可能である。従って、プログラムの方法そのものを変えなくてはならない。

巨大なプログラムには、必然的にバグ (bug)[1]が多く、その修正が大問題である。また、一人で開発することは不可能で、大人数で開発することになる。昔ながらの効率のみを考えた分かりにくいソースコードでは、修正は困難をきたすし、大人数の開発は不可能である。現代のように巨大なプログラムを開発するときには、何よりも

- 分かりやすいソースコードを書くこと

が求められる。本日は、この分かりやすいソースコードを書く方法を学習すると言っても過言ではない。

後で述べるが、そのために、関数というものを使いプログラムを機能 (部品) 毎に分割する。そうすることにより、ソースコードの再利用が可能となる。部品にすればいろいろなプログラムで利用できるのであ

*独立行政法人　秋田工業高等専門学校　電気情報工学科

[1]プログラムの誤りのこと。bug を日本語にして、虫と言うこともある。プログラムの誤りを直す作業を、バグ取りと言う。

る。要するに、部品、例えばボルトのように、ソースコードを使い回しできるのである。再利用しやすいプログラムを書くことも現代のプログラマーに強く求められている。

2 関数 (サブルーチン)

2.1 単語の意味

ここでは、C 言語の関数について、学習する。関数と言うよりも、サブルーチンと言った方が実際の動作が理解しやすい。C 言語では関数と呼ばれるが、サブルーチンと言う言葉もコンピューター科学の世界では、よく使われるので、ほとんど同義語として理解して欲しい。C 言語の文法を学習する前に、これがどのようなもので、なぜ便利なのか説明しておくことが重要であろう。これが分かれば、関数が分かったも同然である。どんなプログラミング言語でもその文法を理解するよりも、それが必要な理由を理解する方がずっと重要であることを忘れないで欲しい。関数 (サブルーチン) は非常に便利なので、ほとんどのプログラム言語でも実装されている。ここでその内容を理解し、将来、他の言語を学ぶときに応用できるようになることを望む。

少し聞き慣れない単語が出てきたので説明しておく。これらは全て、英語である。関数も、元は function を日本語に直した。元の英語の意味を知っておくことは、その機能の理解に役立つ。それらの意味は、以下の通りである。

routine(ルーチン) 1. 決まってすること、日課。2. いつもの手順、所定の手順。3.(コンピューター) ルーチン $\leq$ プログラム中の所定の機能をはたすひとまとまりの部分

main(メイン) 主な、主要な、中心となる

sub(サブ) 副

function(ファンクション) 1. 機能、働き、作用。2. 関数

通常のプログラムは、サブルーチンとメインルーチンから構成される。諸君が今まで作成してきたプログラムの最初の方に `int main()` と書いてきたはずである。それがメイン関数であり、メインルーチンと呼ばれる者である。その範囲は、括弧 { から } までである。知らないうちに、メイン関数を使ってきているのである。これ以外の関数の作り方を学習する。

2.2 どちらのプログラムが分かりやすいか

関数 (サブルーチン) を使うことのメリットは、なんと言ってもソースコードが分かりやすいと言うことである。ソースコードが分かりやすいので、それを記述するのも簡単といえる。実際に比べてみよう。入力された整数が素数か否か判定するプログラムを例にする。プログラムの動作は、次の通りである。

- 整数の読み込み
  - キーボードから整数を読み込む。
  - 読み込んだ値を表示させる。

  - ただし読み込んだ値が 2 以下ならば、再度、整数を読み込む。

- 素数判定
  - 読み込んだ値を、2〜それ自身の値まで割って、余りを求める。
  - 余りがゼロになれば処理終了で、その値を保持しておく。

- 結果表示
  - 余りがゼロになる最小の値が、読み込んだ値と等しければ、「素数です」と表示する。
  - 素数でなければ、割り切れる最小の値を表示する。

2.2.1 関数 (サブルーチン) を使わない場合

```
#include <stdio.h>

/*================================================================*/
/*   main function                                                */
/*================================================================*/
int main(void){
  int i, test, amari;
  char temp;

  /* ---------- キーボードからデータ入力 ------------------*/

  do{
    printf("\n 調べたい整数を入力してください (2 以上)\n");
    scanf("%d%c",&test,&temp);
    printf("素数の判定をする整数は、%d です。\n",test);
  }while(test<2);

  /* ---------- 素数判定 (割り切れる最小) -----------------*/

  amari = 0;

  i=1;
  do{
    i++;                        /* i のインクリメント (1 増加) */
    amari = test % i;           /* test/i の余りの計算    */
  }while(amari != 0 && i <= test);


  /* ---------- 結果表示 --------------------------------*/

  if(test == i){
    printf("\n%d は素数です !!\n\n",test);
  }else{
    printf("\n%d は素数ではありません !!\n",test);
    printf("割り切れる最小の整数 (>2) は、%d です。\n\n",i);
  }

  return 0;
}
```

2.2.2 関数 (サブルーチン) を使う場合

```
#include <stdio.h>

int key_in(void);
int check_prime(int number);
int print_result(int number, int result);

/*================================================================*/
/*   main function                                                */
/*================================================================*/
int main(void){
  int test, kekka;

  test   = key_in();           /* キーボードからデータ入力  */
  kekka = check_prime(test);   /* 判定 (割り切れる最小)     */
  print_result(test,kekka);    /* 結果表示                  */

  return 0;
}


/* <<<<<<<<<< これ以降は、サブルーチン >>>>>>>>>>>>>>>>>>>>>>>>>>>>>*/

/*================================================================*/
/*   キーボードから整数を読み込む関数                               */
/*================================================================*/
int key_in(void){
  int n;
  char temp;

  do{
    printf("\n 調べたい整数を入力してください (2 以上)\n");
    scanf("%d%c",&n,&temp);
    printf("素数の判定をする整数は、%d です。\n",n);
  }while(n<2);

  return n;
}


/*================================================================*/
/*   素数の判定                                                   */
/*================================================================*/
int check_prime(int number){
  int i,amari;

  amari = 0;

  i=1;
  do{
    i++;                       /* i のインクリメント (1 増加) */
    amari = number % i;        /* number/i の余りの計算    */
  }while(amari != 0 && i <= number);

  return i;
}


/*================================================================*/
/*   結果の表示                                                   */
/*================================================================*/
```

```
int print_result(int number, int result){

  if(number == result){
    printf("\n%d は素数です !!\n\n",number);
  }else{
    printf("\n%d は素数ではありません !!\n",number);
    printf("割り切れる最小の整数 (>2) は、%d です。\n\n",result);
  }

  return 0;
}
```

2.3 大規模プログラムの構造 (自動車を例にして)

コンピューターのプログラムは、皆さんが練習問題で作成する数行のものから、Windows2000 の 4 千万行[2]に及ぶものまで多種多様である。このような長いプログラムを書く場合、サブルーチンという考え方がないと、不可能である。

長いプログラムの書き方の説明に入る前に、アナロジー (analogy 類似) として、自動車を作ることを考がえる。自動車も製品で、プログラマーにとってプログラムがそれに当たる。自動車は、非常に多くの部品から出来上がっている。たとえば、ピストン、シリンダー、ボルト、プラグ、タイヤ、ハンドル、ばね、電線、電球、集積回路、コンデンサー、火薬、磁石 $\cdots$ など、それこそ大変な数の部品から作られている。そして、これを組み合わせて自動車が出来上がる。これを、一人で設計することは到底不可能なことは想像できる。それでは、どうやって設計するかと言うと、多くのエンジニアーで寄ってたかって設計するのである。エンジン、トランスミッション、ブレーキ、居住空間、タイヤ、オーディオ機器、空調機器、操作パネル、サスペンション他のように、部品ごとに設計を担当するのである。重要なことは、部品単位、はっきりと区分けをして設計をすることです。

部品ごと設計を行うが、それぞれの接続に関しては、予め決めておく必要がある。たとえば、エンジンとトランスミッションでは、その接続方法を決めてから、各々設計にかかる。そうしないと、最後に組み合わせるときに、それぞれの部品が接続できないということになりかねない。

部品ごと設計が終われば、すべてを組み合わせて、自動車の全体の設計が完了である。予めそれぞれの部品の接続方法が決められているので、ちゃんとした自動車が出来上がる。大雑把に言うと、自動車の設計は、このようにして行われる。製造もほとんどこれに似ている。部品をつくり、最後に組み立てて完成である。この方法は、大規模なものを作るときに行われる一般的な方法で、重要なことは、

- 目的の機能を果たす機械、ここでは自動車を機能ごとの部品に分ける。
- 各々の部品の接続方法は予め決めておく。

である。このように機能別に分けることにより、自動車の設計・製作は格段に分かりやすくなる。全てのことが分かる人が居なくても、みんなで分担して目的の設計ができるのである。複雑なことができるのは、機能毎に分担したからである。

ここでは、自動車をエンジンやトランスミッション等、1 段階に分けましたが、実際には、更に細かく分ける。エンジンであれば、ピストンやバルブ、シリンダーと分ける。部品によっては、更に細かく機能ごと

[2]http://www.fluidlab.naoe.t.u-tokyo.ac.jp/ minnie/FreeBSD/memo.html#SOURCE

に分けることもある。機能ごとに細かく分けると、一つの部品がかなり単純になることが分かるでしょう。

プログラムでは、この特定の機能ごとの集まりをサブルーチンという。メインルーチンは、プログラム全体を統括しているもの、自動車のフレームみたいなものと考えれば良いでしょう (ちょっと強引か)。

自動車みたいな複雑な機械も、単純な機能の部品に分解できるのである。単純な部品の設計は簡単で、誰でもできるようになる。その代わり、その機能だけ分かる人が寄って集って設計をするのである。大規模プログラムも一緒で、それは機能毎に分割され、大人数で寄って集って、コーディングするのである。これで、プログラムを単純な機能に分割することが分かったと思う。

実際に仕事をする場合でも、細かい分かりやすい作業に分割することはきわめて重要である。アポロ計画が象徴的で、この辺については、授業で話す。

2.4 分かりやすいプログラムを書くには

大量のパーツからできている自動車も機能別の部品に分けることにより、その構造が分かりやすくなる。例えば、エンジン、トランスミッション、サスペンション等、機能毎に説明されると分かりやすい。一つ一つの部品、例えば、ボルト、ベルト、シャフトの説明をされても自動車の全体は分からない。自動車の仕組みを分かるためには、機能毎に理解する方が断然簡単である。

プログラムも一緒である。それを構成する機能の集まりで理解した方が簡単である。Windows の 4 千万行もあるプログラムの各行を説明されても全く分からない。

このことから、機能毎に分けるとプログラムが容易になるとともに、その内容が分かりやすくなることが理解できるであろう。その違いを、図 1 と 2 に示す。このように、機能別に分けることにより、プログラムは分かりやすくなる。そして、機能がはっきりすれば分担してプログラムを書くこともできる。ただ、自動車では部品間の接続方法を決める必要があったのと同じように、プログラムでは処理するデータの受け渡しを決めなくてはならない。

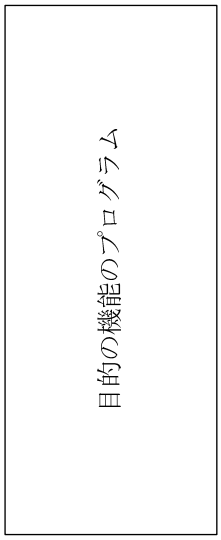

図 1: メインルーチンだけのプログラム

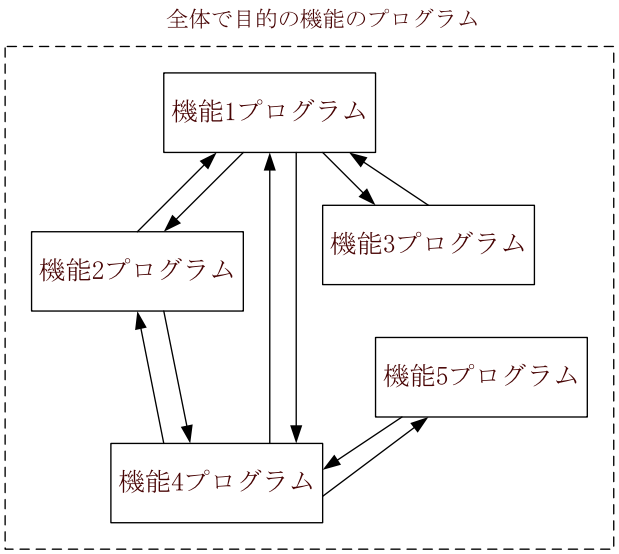

図 2: 機能毎 (サブルーチン) に分割されたプログラム。矢印は、データの流れをあらわす。

自動車の部品にはどれも優劣な無いが、プログラムの場合、1 つだけ特別な機能のルーチンがある。メインルーチン (メイン関数) と呼ばれるものである。これは、最初に実行されるプログラムである。プログラムの場合、1 行毎順番に実行されるため、どれから実行するか決める必要がある。自動車の部品の場合、それぞれが同時に動き機能するため、メインルーチンに対応するものがない。

3 関数 (サブルーチン) の作り方

3.1 関数の書き方の基本

関数を使うためには、以下のようにすれば良い。図 3 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。式は、変数だけでも良い。

  * 括弧無しの方法　return 式;
  * 括弧付きの方法　return(式);

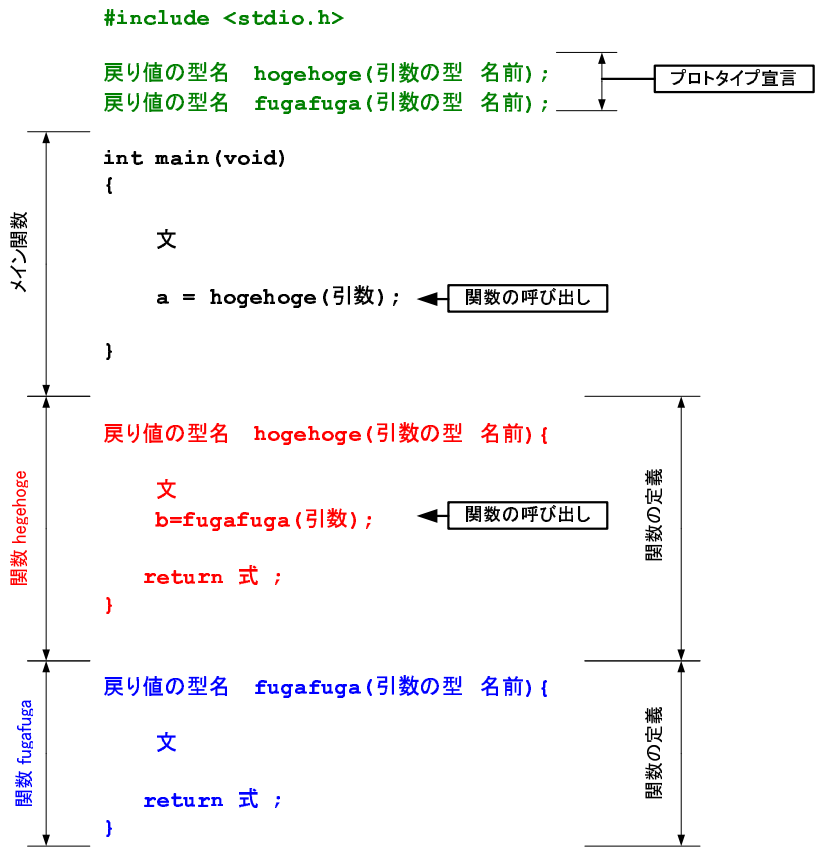

図 3: 関数を使ったプログラムの基本的な書き方

3.2 データの受け渡し

関数を使うときにもっとも気を付けなくてはならないのがデータの受け渡しである。データによって、いろいろな方法があるのでよく理解する必要がある。しかし、ちゃんと理解するためにはポインターが分かる必要があるが、それはまだなので、ここで示すパターンのみを憶えて欲しい。ポインターについては、2 年生で学習する。

以下では、4 つのパターンについて、例を示す。

3.2.1 引数や戻り値が無い場合

この場合、プロトタイプ宣言や関数定義の文では、void(空っぽの) と宣言する。

```
#include <stdio.h>
void hello(void);

/*--------------- メイン関数 ---------------------*/
int main(void){
  int i;

  for(i=0; i<100; i++){
    hello();
  }

  return 0;
}

/*--------------- 関数 ---------------------*/
void hello(void){

  printf("Hello World \n");

}
```

3.2.2 戻り値が一つの場合

```
#include <stdio.h>
double func(double u, double v);

/*--------------- メイン関数 ---------------------*/
int main(void){
  double x, y, z;

  for(x=0; x < 1; x+=0.1){
    for(y=0; y < 1; y+=0.1){
      z=func(x, y);
      printf("%lf\t%lf\t%lf\n",x,y,z);
    }
  }

  return 0;
}

/*--------------- 関数 ---------------------*/
double func(double u, double v){
  double w;

  w = u*u + v*v;

  return w;
}
```

3.2.3 戻り値が複数の場合

```
#include <stdio.h>
void menseki(double a, double *circle, double *square);
```

```
/*--------------- メイン関数 ---------------------*/
int main(void){
  double x;
  double c, s;

  for(x=0; x < 1; x+=0.1){
    menseki(x, &c, &s);
    printf("%lf\t%lf\t%lf\n",x,c,s);
  }

  return 0;
}

/*--------------- 関数 ---------------------*/
void menseki(double a, double *circle, double *square){
  double pi;

  pi=3.141592;

  *circle = pi*a*a;
  *square = 4.0*a*a;
}
```

3.2.4 配列の場合

```
#include <stdio.h>
void twice(int n,int a[32][32]);

/*--------------- メイン関数 ---------------------*/
int main(void){
  int a[32][32];
  int i,j;

  for(i=0; i < 32; i++){
    for(j=0; j < 32; j++){
      a[i][j]=i+j;
    }
  }

  twice(32, a);

  return 0;
}

/*--------------- 関数 ---------------------*/
void twice(int n, int a[32][32]){

  int i,j;

  for(i=0; i < n; i++){
    for(j=0; j < n; j++){
      a[i][j] = 2*a[i][j];
    }
  }
}
```