C 言語のサンプルプログラム (4)

計算機応用　5E 2003.05.15

１．関数

1.1 関数とは何か

FORTRAN には、関数副プログラムやサブルーチン副プログラムを用いて、プログラムを分割し、処理の内容を分かり易く記述します。C 言語の場合、それに対応するものが関数です。複雑で大掛かりなものを、小さい機能に処理を分けるために C 言語の関数があります。大きな処理を記述するより、処理を小さく分けるほうが、はるかに分かりやすく、デバックも簡単です。

関数の中で特殊なものが main 関数で、プログラムの実行の開始を宣言します。これが無いと、どこからプログラムを実行してよいか、分かりません。今まで、使ってきた printf も関数です。括弧内に引数、データを与え、処理を実行します。その他、数学関数などいろいろな関数があったと思います。

C 言語では、通常の変数を関数に渡す場合、呼ばれた関数は、アドレスではなく、各引数の自分用の一時的なコピーを受け取ります(call by value)。これは、呼ばれた関数が呼び出しを行った元の関数の引数を変えることができないことを意味しています。各引数は局所変数で、関数の独立性が非常に強い言語です。

一方、FORTRAN の場合は、SUBROUTINE にはそのアドレスが渡され、元の引数を変えることができます(call by reference)。この点が、引数を渡す場合の FORTRAN と C 言語の大きな違いです。とはいっても、C 言語でもアドレスを渡すことはできます。引数にポインターを指定すればよいのです。この辺の詳細については、サンプルプログラムで示します。

一方、配列名が引数として渡されるときは、その配列の最初のアドレスが渡されます。これは、配列をすべてコピーすると、かなり処理が必要なためです。そのため、配列の場合はアドレスが渡されます。

関数がそれが呼ばれた関数に値を引き渡すには、普通、返り値を使います。しかし、これは２個以上の変数の値を引き渡すことができません。それを避けるには、引き渡すべき変数を外部変数として定義しておく方法もありますが、これは関数の独立性を損なうので、勧められません。グローバル変数ではなく、ポインターを使えば、2 個以上の変数の値を返すことができます(call by reference)。FORTRAN と同じです。

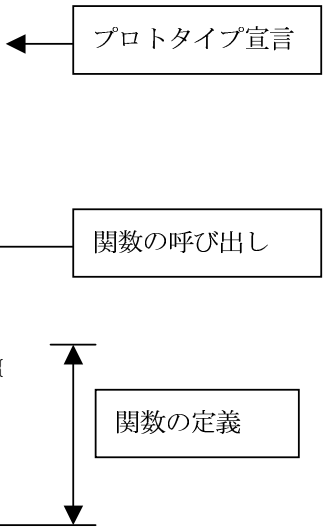
あと C 言語の関数の特徴は、プロトタイプ宣言です。関数の引数と戻り値の型を宣言します。これにより、コンパイル時に引数のチェックを行います。

今まで使ってきた関数は、main 関数からコールされましたが、他の関数からもコールできます。それどころか、自分自身をコールすることもできます。自分自身を再帰的にコールする場合、無限ループに入る可能性がありますので、それを避ける処理を記述しなくてはなりません。

1.2 関数の一般形

一般的に関数は、次のような形式で使います。関数名は、hogehoge です。

```
#include <stdio.h>

戻り値の型名  hogehoge(引数の型と名前);

main()
{

   文

   a = hogehoge(引数);

}

記憶クラス  型名  hogehoge(引数の型と名前){

   文

  return(式);
}
```



ここで、hogehoge という関数は、2 行目の

戻り値の型名 hogehoge(引数の型と名前)

でプロトタイプ宣言をしています。コンパイル時の引数のチェックに使うため、それらの型を宣言しています。それらの代表的な型を表 1 に示します。通常は、int と double しか使いません。これらの int や double も、表 1 の char のように、ポインターを示しても良いです。C 言語サンプルプログラム(3)で示したデータの型と同じです。

引数や戻り値が無い場合、void を使います。これは、よく使いますので、覚えておく必要があります。void とは「空の」と言う意味です。

表 1 代表的な型。これ以外にもある(データの型を参照)。

型	戻り値	引数	備考
int	int 関数名(引数)	型 関数名(int 名前)	整数型
double	double 関数名(引数)	型 関数名(double 名前)	倍精度実数型
char	char 関数名(引数)	型 関数名(char 名前)	文字型(1 文字)
char	char *関数名(引数)	型 関数名(char *名前)	文字列
void	void 関数名(引数)	型 関数名(void)	戻り値や引数無し

関数の定義は、

記憶クラス 戻り値の型名 hogehoge(引数の型と名前)

から始まります。記憶クラスは、表 2 の通りです。通常は extern なので、デフォルトとなっています。したがって、記憶クラスを書くことは無いです。非常に大きなプログラムを開発する場合、多くのファイルに分割します。そのときに static を使います。

戻り値の型名と引数の型と名前は、プロトタイプ宣言と同じです。プロトタイプ宣言をコピ

ーペーストしましょう。ミスが減ります。

表 2 記憶クラス

記憶クラス	機能	備考
extern	その関数は他のファイルから参照可能	こちらがデフォルト。 通常は、書かなくてよい。
static	その関数は他のファイルから参照不可能	プログラムのファイルを分割した場合、そのファイルでしか使わない関数に適用する。

関数の戻り値は、

```
return(式);
```

で決まります。この式の値が戻り値になります。

関数の呼び出しと戻り値の代入を

```
a = hogehoge(引数);
```

で行っています。プロトタイプで宣言された型の変数を引数として渡せば、先ほど述べた関数の定義どおりに計算して、戻り値を返します。

1.3 値渡しとアドレス渡し

C 言語で関数にデータを渡す一般的な方法は、値渡し(call by value)です。これは、呼び出し側で実引数[1]に値をセットして関数をコールすると、関数側の仮引数[2]に、その値をコピーして処理が行われます。そのため、関数側で処理する仮引数の値が、実引数に影響を及ぼすことはありません。関数の独立性を維持するために、非常に良い方法です。

FORTRAN の場合は、アドレス渡し(call by reference)です。これは、実引数に値をセットしているように見えますが、値をセットしたメモリー領域を仮引数に渡しています。したがって、関数側で処理する仮引数の値が、実引数に影響を及ぼします。

次の 3 個のプログラムを実行して、値渡しアドレス渡しの違いを理解してください。

サンプルプログラム(swapf.f)

まず、アドレス渡しのフォートランです。サブルーチン swap で a と b の値を入れ替えます。これは、アドレス渡しです。

```
c ----- Main routine --------
      integer a, b

      a=1
      b=-1

      call swap(a,b)
      write(6,*)a,b

      stop
      end

c ----- Subroutine ---------
      subroutine swap(a,b)
      integer a, b, c

      c=a

      a=b
      b=c

      return
      end
```

コンパイル方法

```
f77 -o swapf swapf.f
```

実行結果

実行結果は、以下の通りです。メインで代入された値が、subroutine swap で交換されていることが分かるでしょう。

```
 -1 1
```

---

1 呼び出し側の引数のこと。

2 呼び出された側の関数の引数のこと。

サンプルプログラム(swapc.c)

次は、値渡しのＣ言語です。関数 swap で a と b の値を入れ替えます。これは、実引数の値のコピーである仮引数を入れ替えているだけです。もとの実引数には、影響与えませんので、main に戻っても、値は変化しません。

```
#include <stdio.h>
void swap(int a, int b);

/* ----- main funtion --------  */
main()
{
       int a, b;

       a=1;
       b=-1;

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

}

/* ----- swap function --------- */
void swap(int a, int b)
{
       int c;

       c=a;

       a=b;
       b=c;

}
```

実行結果

実行結果は、以下の通りです。関数 swap で値を変えたつもりですが、main 関数では変化はありません。仮引数の値を入れ替えても、実引数は変化しません。

```
 1  -1
```

サンプルプログラム(swapcp.c)

次は、C 言語でポインターを使ってアドレス渡しの場合です。関数 swap に a と b のアドレスを渡し、値を入れ替えます。実引数のアドレスを渡して、そのアドレスの値を操作しています。したがって、main 関数の実引数に、影響与えます。

```
#include <stdio.h>
void swap(int *a, int *b);

/* ----- main funtion --------  */
main()
{
       int a, b;

       a=1;
       b=-1;

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

}

/* ----- swap function ---------- */
void swap(int *a, int *b)
{
       int c;

       c=*a;

       *a=*b;
       *b=c;

}
```

実行結果

実行結果は、以下の通りです。関数 swap では、実引数のあどれるの値を操作しています。したがって、main 関数に戻った場合、その値が入れ替わっています。

```
 -1  1
```

1.4 戻り値が無い場合

戻り値が無い関数の例を示します。

サンプルプログラム(noreturn.c)

戻り値の無い関数 printsin を定義して、使っています。あわせて、コンソールからデータを読み込むプログラムの練習にもなっています。

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

void printsin(double x);

/* ------- main -------------------------- */
main()
{
   char a, b;
   double z;

   while(1){
      printf("next value ? (y/n)  ");
      scanf("%c%c", &a, &b);

      if(a=='y'){
         printf("value sin(z) z = ");
         scanf("%lf%c", &z, &b);
         printsin(z);
      }else if(a=='n'){
         break;
      }else{
         printf("input should be y or n\n");
      }
   }
}

/* ------ printsin function -----------------*/
void printsin(double x)
{

   printf("sin(%lf)=%lf\n",x,sin(x));

}
```

コンパイル方法

数学関数 sin を使っているので、 math.h をインクルードしています。そのため、コンパイルオプションに -lm が必要です。

```
cc -lm -o noreturn noreturn.c
```

1.5 戻り値が 1 変数の場合

戻り値が 1 つの場合の関数の例を示します。値渡しで関数にデータを送り、値渡しで関数からデータを受け取っています。関数が完全に独立しているため、思わぬところでデータの変更が発生しなので、最も良い方法です。

サンプルプログラム(onereturn.c)

戻り値が倍精度実数の mysin という関数を定義して使っています。mysin は、正弦関数をテイラー展開の 3 項まで取っています。

pow(x,y)は、x^y を計算する関数です。

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

double mysin(double theta);

main()
{
   double x, y, z;
   double dpi, pi=4*atan(1);
   int i;

   dpi = pi/2/100;

   printf("=== theta    mysin    sin   ===== \n");

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

      x = dpi*i;
      y = mysin(x);
      z = sin(x);

      printf("%lf   %lf   %lf \n", x, y, z);
   }
}

/* ================================================ */
/*  taylor expansion of sin function             */
/* ================================================ */
double mysin(double theta)
{
   double yy;

   yy = theta-pow(theta,3)/6+pow(theta,5)/120;

   return(yy);

}
```

実行結果

sin 関数とそのテイラー展開の 3 項までの値を表示しています。非常に収束が良いです。

1.6 戻り値が複数の場合

return 文を見ると、複数の戻り値を返すことが不可能であることが分かりす。FORTRAN では、それが可能です。FORTRAN と同じように、アドレス渡しにすれば良いのです。

実際には、引数をポインターにして、呼び出し側と関数側でデータを共有します。共有した複数のデータを操作することにより、複数の戻り値を返したようにするのです。p.7 のサンプルプログラム swapcp.c が、まさにその例です。

サンプルプログラム(fourreturn.c)

正弦関数の和と差、積、商を計算します。ポインターを用いて、4 つの戻り値を得ています。

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

void sincal(double a1, double a2,
       double *y1, double *y2, double *y3, double *y4);

/* ========================================================= */
/*  main                                    */
/* ========================================================= */
main()
{
   double z1, z2, wa, sa, seki, shou;
   double pi = 4*atan(1);

   z1 = pi/3;
   z2 = pi/6;

   sincal(z1, z2, &wa, &sa, &seki, &shou);

   printf("%lf  %lf  %lf  %lf\n",wa, sa, seki, shou);

}

/* ========================================================= */
/*  function  sincal                                        */
/* ========================================================= */
void sincal(double a1, double a2,
       double *y1, double *y2, double *y3, double *y4)
{
   double x1, x2;

   x1 = sin(a1);
   x2 = sin(a2);

   *y1 = x1+x2;
   *y2 = x1-x2;
   *y3 = x1*x2;
   *y4 = x1/x2;

}
```

実行結果

```
1.366025  0.366025  0.433013  1.732051
```

1.7 配列の受け渡し

配列の受け渡しは、通常の 1 変数の場合と異なります。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す。しかし、驚いたことに、配列の場合、実引数のアドレスが、仮引数に渡されるのです＼(°o° ;) /。FORTRAN と同じです。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、そのサイズは大きい場合が多い。大きな配列をコピーして刈り引数に渡すと、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である。したがって、配列のコピーは不経済なので、避けるべきである。

ということが理由です。仕様でそのようになっているので仕方がありませんが、不統一の感はぬぐえません＼(`o''') 。この辺が、C 言語は難しいものと思う人が出てくる原因かもしれません。

サンプルプログラム(transpose.c)

関数 transpose で行列を転置しています。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ます。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します。

先頭アドレスは、transpose 関数では、

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

の形で受け取ります。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ますが、後々の操作が大変になります。したがって、配列を受け取る場合は、配列として受け取ってください。

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ません。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ます。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のです。もちろん書いても良く、x[5][5]でも問題はありません。

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す。したがって、x[5][5]のようにすべて記述しなくてはなりません。

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])と書きます。左端のみ省略可です。

```
#include <stdio.h>
void transpose(int x[][5]);

/* ----------------- main ----------------------*/
main()
{
   int i;
   int a[5][5]
     ={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};

   for(i=0; i<=4; i++){
      printf("%d %d %d %d %d \n",
     a[i][0], a[i][1], a[i][2], a[i][3], a[i][4]);
   }

   transpose(a);
   printf("\n");

   for(i=0; i<=4; i++){
      printf("%d %d %d %d %d \n",
     a[i][0], a[i][1], a[i][2], a[i][3], a[i][4]);
   }

}

/* -------------- transpose function -------------*/
void transpose(int x[][5])
{
   int i, j, temp;

   for(i=0; i<=3; i++){
      for(j=i+1; j<=4; j++){
         temp = x[i][j];
         x[i][j] = x[j][i];
         x[j][i] = temp;
      }
   }
}
```

実行結果

```
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
```