ソフトゼミA　第６回 関数

■　はじめに

今まで、printf や scanf など、予め用意されていた関数を使ってきました。これら標準で用意されている関数を作ることは(特に入出力系は)とても難しいのですが、関数は自作することができます。というわけで、今回は自分で関数を定義して使っていく方法について学びましょう。

■　関数とは

C 言語での「関数」は、処理の途中で呼び出すことによって、定義されている「ある一連の処理」をするものです。関数を呼び出すには、関数名と関数に渡す数(引数, ひきすう)を記述します。

関数の定義は、**他の関数の外で**以下のように書きます。引数を定義する際、同じ型の引数が複数並んでいても、すべての引数の型名を省略せずに記述しなければなりません。(引数なしなら void と書く)

```
①戻り値の型 ②関数名(③引数の型 引数名 1, 引数の型 引数名 2, …) {
        …関数の処理
        return 値(式でも可);                    //1 つの値しか返せないので注意すること
}
```

例えば、横の長さ w, 縦の長さ h の長方形の面積を求める関数は次のように書けます。(例として rectangle _s という名前をつけてあります。)

```
double rectangle _s ( double w, double h ){
   double s = w * h;
   return s;
}
```

関数の呼出しは以下の様に書きます。関数の中身を理解していなくても簡単に呼び出せるのでライブラリ関数(printf 等)も簡単に扱えます。

```
関数名(引数 1 , 引数 2, …);
```

関数の中で宣言された変数の有効範囲はその関数の中だけです。さっきの rectangle _s で宣言されている変数 s は他の関数では使用できません。(別の関数で同じ名前の変数を宣言することはできますが、全く別の変数として扱われます。このように、ある関数の中で宣言されている変数のことをその関数の「ローカル変数」と言います。

これとは逆に、関数の外で変数を定義することができます。この変数を「グローバル変数」と呼び、そのソースファイルのどの関数からでも参照できます。しかし、どの関数からでも参照できるということは、どの関数からでも書き換えができてしまうということでもあり、バグの温床となりやすいので、極力使わないようにするべきです。なお、ある関数内でグローバル変数とローカル変数で同じ名前の変数が存在する場合は、その名前を参照するとローカル変数の値が出てきます。

先程例に挙げた rectangle _s を使って、一つ例を挙げてみます。

a06_1.c

```
#include <stdio.h>
double rectangle_s ( double w, double h ){ /*  ここでの w, h は「仮引数」と呼ぶ  */
   double s = w * h;
   return s;
}
int main( void ){
   double width, height, s;
   scanf( “%lf”, &width );
   scanf( “%lf”, &height );

   /*  関数「rectangle_s」を呼び出す。ここでの width, height は「実引数」と呼ぶ。  */
   s = rectangle_s(width, height);

   printf( “%f¥n”, s);
   return 0;
}
```

s = rectangle_s (width, height); の行が実行される時、次のように処理されます。

1. 関数 rectangle_s が呼び出される。main 文側の引数(実引数といいます。) width, height はそれぞれ関数 rectangle_s 側の引数（仮引数といいます。）である w, h に代入される。
2. 関数 rectangle_s のローカル変数「s」に w * h の結果が代入される。
3. 関数 rectangle_s が戻り値「s」を main 文に返す。
4. main 文のローカル変数 s に返ってきた値が代入される。

既に気づいている方も多いとは思いますが、今まで書いてきた int main(void)～というのも main という名前の引数無し、戻り値 int 型の関数です。main という名前をつけるとプログラムが開始した直後に呼ばれるようになっています。余談ですが、main 文の最後に書いている「return 0;」は「0」を戻り値とすることで、そのプログラムが正常に終了したということを知らせるために書くことになっているものです。

■ 具体的な関数の使用例

➢ 値を返す関数

以下のプログラムは、実数 r を入力することによって、r を半径とする球の体積を求めるものです。

a06_2.c

```
#include <stdio.h>
double pi( void ){                          /*引数なしのとき(void)と書く*/
  return 3.141592654 ;                      /*戻り値の指定*/
}
double sphere_v( double r ){                /*この()内の r を仮引数という*/
  double v;                                 /*この v は関数内でのみ有効…ローカル変数*/
  v = (4*pi()*r*r*r) / 3;                   /*戻り値のある関数は式中で呼び出せる*/
  return v;                                 /*()内は式でもいい、後()は書かなくてもいい*/
}
int main( void ){
  double r, v;                              /*関数が違えば同じ名の変数でも別物*/
  printf( "半径 r = " );   scanf( "%lf", &r );   /*入力…double 型は%lf と書くことに注意*/
  v = sphere_v(r);                          /*計算…この()内の r を実引数という*/
  printf("球体の体積 v = %.2f¥n", v);     /*出力…桁数の指定*/
  return 0;
}
```

※printf 内にある%.2f は double 型の実数を小数第 2 位まで表示する書式指定です。

※球の体積 $V = (4/3) \times \pi r^3$ （この公式は積分により導出される。）

※別に double pi(void)はグローバル変数 pi でもいいかもしれませんが、説明のために関数にしています。

➢　値を返さない関数

次のプログラムは、第 4 章の「a04_2.c」で出てきた「100 が入力されるまで無限ループするプログラム」を書き換えたものです。

a06_3.c

```
#include <stdio.h>
void wait( void ){                          /*戻り値がない…void 型の関数*/
  int input;
  printf( “wait 関数が呼び出されました。¥n” ):
  for(;;){                                  /*無限ループ。 while(1)でもよい。*/
    printf( "100 が入力されたら wait 関数を脱出します。¥n" );
    scanf(“%d”, &input);
    if(input ==100 ){
      printf( “wait 関数を脱出します。¥n” ):
      return;                               /*別に break;でも構わないけれど… */
    }
  }
}
int main(void){
  wait();                                   /*特定の数字の入力を待つ関数*/
  printf( “main 文に戻ってきました。終了します。¥n” ):
  return(0);
}
```

※先ほどとの違い…戻り値の型を void にしただけ。

このように、戻り値がない関数もつくることができます。このような関数は、単に呼び出すだけで、関数の結果を変数に代入したり、計算に使ったりすることができません。このような関数で、関数の処理を途中で打ち切る場合には「return;」とだけ書きます。return;は省略可能で、書かれていない場合は、関数の最後まで処理が進むと終了します。上の例では、100 が入力された時に、return;を使って関数から抜けていますが、別に今まで通り無限ループを抜ける「break;」を使っても構いません。return;を使った場合には、その場で関数の処理が打ち切りになりますが、break;を使った場合には、関数の最後まで到達した時点で自動的に関数の処理が終了します。

■　注意すべき点

➢　関数の順番

今まで、関数は全て main 文の上で定義してきました。main 関数の下で自作の関数を定義してもかまわないのですが、コンパイラは上から順番にプログラムを読んでいき、未知の関数がでてきたら戻り値の型を勝手に int 型とみなしてコンパイルしようとするので、その結果エラーが出る可能性があります。そこで関数のプロトタイプ宣言というものを行えば、関数間の呼出しを自在にできるようになるので便利です。

※　プロトタイプ宣言…後で出てくる関数の受け取る引数の型、返す値の型などをコンパイラに形だけ伝えておく。(宣言だけの場合コンパイルエラーが出る)

プロトタイプ宣言は、以下のように関数の「戻り値の型」「名前」「仮引数」のみを書いてセミコロンを打って終了します。

```
#include <stdio.h>
int a(int x, int y);   /*プロトタイプ宣言*/
int main(void){      /*main 関数が見やすくなるかも*/
  …略
}
int a(int x, int y){   /*プロトタイプ宣言した関数は後で必ず定義される必要がある*/
  …略
}
```

ゼミ B では、規模の大きいプログラムを作るので、プロトタイプ宣言を行なってプログラムを見やすくすることをします。

➢　引数について

関数で複数の値を返したいというケースに当たることがあります。しかし、関数は 1 つの値しか返すことができません。現時点では 2 つ以上の値を返すことは困難なのですが、次回学習するポインタの応用によって、擬似的に 2 つ以上の値を返すことができるようになります。

■　関数を使用するメリット

・同じ仕事を何度も繰り返す場合に、同じプログラムを何度も書く必要がなくなります。間違った部分の修正も容易になります。

・複雑なプログラムも、決まった仕事をする部分を関数として抜き出し独立に理解すれば、理解すべき大きさも小さくなり、流れを掴み易くなります。(プログラムの可読性が上がる。)

■ 練習問題

今回の問題は関数を作る問題ですが、必ず main 文も書いて関数の動作を確認すること！

1. ある数の絶対値を返す関数 double get_abs(double x)を作れ。
2. ある正の整数 a から b までの n の倍数を全て表示する関数 void display_numbers(int a, int b, int n)を作れ。
3. a06_2.c を改造して、半径 r の球の表面積を求める関数 double sphere_s(double r) を作れ。ただし、球の表面積 $S = 4\pi r^2$ である。

■ 発展（余裕がある方向け）

➢ static 指定子

a06_4.c

```
#include<stdio.h>
void fn( void ){
   int a=0;
   static int b=0;                    /*関数の実行が終わっても値が破棄されない*/
   a++;   b++;                        /*つまり b が初期化されるのは最初に呼出された時だけ*/
   printf("a=%d, b=%d¥n",a,b);
}
int main( void ){
   int i;
   for(i=0;i<10;i++){
      fn();
   }
   return(0);
}
```

static というキーワードを宣言時につけられたある関数のローカル変数は、その関数が終了した後も破棄されず、次に同じ関数が呼び出された時に初期化されずに残り続けます。

結果はコンパイルして実行して確かめてみてください。

➢ 再起呼び出し

a06_5.c

```
#include <stdio.h>
int fact( int n ){
   if( n == 0 ){ return(1); }       // 0! = 1
   return( n * fact(n - 1) );       // n >= 1 のとき  n! = n * (n-1)!
}
int main(void){
   int i, ans;
   printf("数値の入力:");   scanf("%d", &i);
   ans = fact( i );
   printf("%d の階乗は%d¥n", i, ans);
   return 0;
}
```

※int 型で表せる数字の限界を超える(オーバーフローする)と、値は正確ではなくなります。

ある関数(この例では fact)が定義されていて、fact の中で fact 自身を呼び出すことができます。これを関数の再起呼び出しといいます。再起呼び出しの特徴としては、fact の中で呼び出された fact は、呼び出し元とは別の fact であるということです。ローカル変数がある場合には、呼び出し元の関数のローカル変数と、呼び出された関数のローカル変数は別物として扱われます。今回の例では、main 文から実引数 i で fact が呼び出されて、i が 0 でないなら i に fact(i-1)の結果をかけたものを返します。ここで、fact(i-1)は i-1 が 0 でないなら、i-1 に fact(i-2)の結果をかけたもので…という風に n が 0 になるまで呼び出し続けます。n が 0 になったら 1 を返し、返ってきた fact(0)の結果に 1 がかかって、fact(1)の結果に 2 がかかって…という風に、値が戻っていき、最終的には main 文に i!の値が返ってきます。

再帰呼び出しはこのように関数を美しく書くことはできるのですが、メモリをよく消費するなどの欠点があり、無限ループに陥りやすいなどの難しさがあります。用法・容量を守って正しくお使いください。