

## 第6回 関数

今回は第1回で少しだけ述べた関数について学んでいきます。関数は、プログラムの塊、処理の塊です。複数の処理を纏め、これを関数とし、使用したい時には関数名と定められた記法に従って関数を呼び出すことで、まとめられた複数の処理を一行で使用することができます。これにより、同じ処理をしたい時にいちいち同じことを何度も各必要が無くなり、プログラムが猥雑になるのを防ぐことができます。

C 言語には、標準で用意された素晴らしい関数群(標準ライブラリ)がたくさんあるため、プログラミングが効率よくできるようになっています。関数は、ユーザが自ら新たな関数を作ることが出来ますので、見易さや機能性においても、有利にプログラミングが出来ます。

### ○関数の定義と呼び出し

関数には「関数名」、「引数(ひきすう)」、「戻り値」の3つの要素があります。数学の関数と比較してみましょう。

$f(x)=x^2$  という関数があったとします。この時、C 言語ならば「 $f$ 」が関数名、 $x$  に代入される実数が引数、 $x$  に実際に値を代入した時に算出される  $x^2$  の実際の値( $x=2$  ならば  $4$ )が戻り値です。以下のプログラムを見てください。

```
#include<stdio.h>
int add( int x , int y ){ /*関数の定義( )内の x、y を仮引数という*/
    int z ;
    z = x + y ;
    return ( z ) ; /*戻り値の指定*/
}
int main( void ){
    int a , b , ans ;
    printf( "a=" ) ;
    scanf( "%d" , &a ) ;
    printf( "b=" ) ;
    scanf( "%d" , &b ) ;
    ans = add( a , b ) ; /*関数 wa の呼び出し &ans へ wa の結果を代入*/
    /*ここでの a、b の実際の値実引数という*/
    printf( "a+b=%d です\n" , ans ) ;
    return ( 0 ) ;
}
```

main 関数の上にある `int Add( 中略 ){ 中略 }` が自分で定義した関数です。関数「add」は二つの `int` 型の変数を受け取り、その和の値を戻り値として返す関数です。数学の関数で表すならば、 $add(x, y)=x+y$  です。C 言語においてはこの時、`add` は関数名、`int x`、`int y` はそれぞれ引数の型と引数の名前(またはそれら2つをまとめて仮引数)、`add` の前の `int` が戻り値の型です。関数の呼び出し時に、`x`、`y` に渡される実際の値が引数、`x`、`y` の和として、`add` から `return` される実際の値が戻り値です。

関数名は関数につけられた名前です。関数名も変数名と同様に使用できる文字に制限があります。通例 C 言語における関数名は小文字と数字だけで構成されることが多いです。

引数は関数に対して渡す情報です。`printf` 関数や `scanf` 関数を使用する際に `( )` の中に書いたものが引数です。関数内部の処理で引数が使用されます。引数として渡された数値は、渡された関数内でローカル変数のように振舞います。渡された側の関数内でこれに代入しても、ローカル変数ですので、渡した側の関数の変数には影響はありません。関数内で仮引数がローカル変数として宣言され、呼び出し時に `( )` の中に記述した値が初期値として代入される、と考えるとわかりやすいでしょう。

戻り値は関数内の処理が終わった時関数側から返される情報です。関数内の `return` 文によって返されます。main 関数内の最後の「`return ( 0 ) ;`」は戻り値として `0`(エラー無しのサイン)を返す、という意味です。

関数の定義は次のような形式で記述します。

```
戻り値の型 関数名( 引数の型 引数の名前, ... ){
    関数内部の処理 ;
    ...
}
```

また、関数の呼び出しは、次のような形式で記述します。

```
関数名( 引数 1 , 引数 2 , ... )
```

関数を呼び出すだけで、戻り値を利用しない場合は上記のように記述後、セミコロンで OK です。戻り値を利用する場合は、式中に関数の呼び出しがあれば、その部分が戻り値で置き換えられたと考えれば良いので、あとは様々に演算子を利用して煮るなり焼くなり好きに利用しましょう。戻り値や引数(≠仮引数)には自由に式(変数または実数を演算子でつないだもの)をいれることができます。これを利用すれば関数 `add` は以下のように置き換え可能です。

```
int add( int x , int y ){
    return ( x + y ) ;
}
```

戻り値や引数を利用しない場合、`void` を記述するか、何も記述しません。C 言語の規格上、厳密には両者の意味は違うので、C 言語においては `void` と明記することが望ましいと言えるでしょう。以下のように記述します。

```
void output( void ){
    printf( "ソフトゼミ A 関数\n" ) ;
}
```

この関数は、『ソフトゼミA 関数』という表示を行うだけあって、特に戻り値を返す必要がありません。`void` とは、『空』という意味です。`void output(void)` の最初の `void` は戻り値がない、といった感じの意味で、戻り値がない場合 `return` は省略することができます。`return` を記述した場合はその時点でその関数内の処理は打ち切られます。これは戻り値が存在する場合も同様ですが、戻り値が `void` の関数の場合は `return` の後に式を付ける必要がありません。二つ目の `void` は引数を受け取らないという意味になります。引数が `void` の関数を呼び出す場合は、

```
output() ;
```

のように記述します。こちらは関数であることを明示するため、`()` が必要になりますが、その間には空白すら必要ありません。

## ○プロトタイプ宣言

前回の構造体で、構造体変数の宣言より前にその構造体の定義が必要であったのと同様に、関数においてもその関数の呼び出しよりも前に定義が必要となります。そのため、先程のプログラムにおいては `main` 関数よりも前に関数 `add` を記述する必要がありました。これを簡便にしてくれるのがプロトタイプ宣言です。

通常、関数の呼び出しが定義より前にあると「そのような関数は知らない」といった趣旨のエラーをコンパイラが吐きますが、以下のようにすることで、コンパイラに関数の存在を伝えることができます。

```
戻り値の型 関数名( 引数の型 , 引数の型 , ... ) ;
```

プロトタイプ宣言は関数の呼出よりも前にプログラム内にその関数が存在することをコンパイラに伝えるものなので、関数の呼び出しよりも先に記述する必要があります。また、この時引数の名前は必要ありません。

## ○ヘッダとインクルード

プロトタイプ宣言によって、関数の引数や戻り値の定義が与えられると、その関数を安心して呼び出せることがわかりましたね。それでは今まで謎の存在だった `#include <stdio.h>` に触れていきましょう。

実は、今まで何度も利用してきた `printf` 関数や `scanf` 関数は、`stdio.h` というファイルの中でプロトタイプ宣言されていたのです。`#include` はそのファイルの中身をここに挿入する、という「プリプロセッサディレクティブ」です。プリプロセッサディレクティブはプリプロセスを行わせる命令のことです。プリプロセスとは前処理、という意味で、プログラムをコンパイルする時、その前にソースコードに処理を行う事です(実際にソースコードが改変されるわけではない)。プリプロセッサディレクティブは `#` で始まり、セミコロンが無いのが特徴です。

`stdio.h` はヘッダと呼ばれるファイルです。`.h` という拡張子はその略です。ヘッダには主に関数のプロトタイプ宣

言が含まれており、C 言語に標準で備わる様々な関数を利用する際、`#include <ファイル名.h>`と記述することで使用可能となります。また、`<>`のかわりに`""`で括ることで、自作ヘッダを利用することができます。ソースコードと同じフォルダ(ディレクトリ)にヘッダをおいておけば`#include "ファイル名.h"`とすることでインクルードできます。

### ○グローバル変数

前回少しだけ触れたグローバル変数(大域変数)は、メイン関数の外で定義することにより、どの関数からでも呼び出しができ、代入が可能で言葉どおり、「グローバル」な、広い範囲で変数が有効になることを意味しています。

```
#include <stdio.h>
int x ; /*グローバル変数として x の宣言*/
void f( void ){
    x = 10 ; /*グローバル変数 x への代入*/
}
int main( void ){
    x = 1 ; /*グローバル変数 x への代入*/
    printf( "x=%d\n" , x ) ; /*x=1 が表示される*/
    f() ; /*x=10 が代入される*/
    printf( "x=%d\n" , x ) ; /*x=10 が表示される*/
    return ( 0 ) ;
}
```

上記の例ではまずグローバル変数 `x` を宣言し、`main` 文の内で `x` に 1 を代入しそれを表示します。その後、関数 `f` で `x` に 10 を代入し表示します。結果、画面には、

```
x=1
x=10
```

と表示されます。関数内で宣言された変数はローカル変数(局所変数)と呼ばれます。ローカル変数は宣言された関数内でのみ使えます。そのため複数の関数で同じ変数名の変数を宣言することができ、それらは互いに影響を及ぼすことはありません。また、グローバル変数と局所変数でも同じ変数名を使うことができます。その場合はグローバル変数よりローカル変数が優先されます。以下に例を示します。

```
#include<stdio.h>
int x ; /*グローバル変数として x の宣言*/
void f( void ){
    x = 10 ; /*グローバル変数 x の初期化*/
}
int main( void ){
    int x ; /*←ここに局所変数として x の宣言を追加した*/
    x = 1 ; /*ローカル変数への代入。グローバル変数はまだ初期化されていない。*/
    printf( "x=%d\n" , x ) ; /*ローカル変数 x の値を出力*/
    f() ; /*ここでグローバル変数 x の初期化*/
    printf( "x=%d\n" , x ) ; /*ローカル変数 x の値を出力*/
    return ( 0 ) ; /*プログラム終了のお知らせ*/
}
```

結果は

```
x=1
x=1
```

となります。

### ○演習問題

`int` 型の 2 変数を受け取り、加算した値と乗算した値を計算する関数をそれぞれ作り、2 変数をキーボードから入力し、計算結果を出力する `main` 関数を記述せよ。