関数

Ｃ言語のプログラムは関数を利用しながら作成する。ここまで printf や scanf を習ったが、これらは stdio.h に用意されている標準入出力関数である。いままで特に意識をしてこなかったと思うが、これらは stdio.h をインクルードし、コンパイル、リンクするときに標準入出力に関するオブジェクトファイルを自分が書いたプログラムと結合することで利用している。つまり、関数にはその動作を記述しているプログラムが必要となる。この関数は自分で定義して利用することもできる。今回は関数の定義とその使い方について説明していく。

関数の役割

関数を用いてプログラムを作成するメリットは、プログラムが処理ごとにまとめることができるので部品のように扱うことが可能となり、一度作成してしまえば、次にはその関数を利用するだけで済むので、プログラム作成の手間が減る。また、処理ごとの集まりができるので、プログラムが読みやすくなる。これらのメリットはプログラムが大規模になればなるほど大きい。

簡単な関数を用いたプログラム

それでは簡単な例を用いて、関数について説明していく。以下のプログラムは２つの整数値の和を求め、出力するプログラムである。

```
#include <stdio.h>

int main(void)
{
        int a = 3, b = 4, c;
        c = a + b;
        printf("a+b=%d¥n", c);
        return 0;
}
```

出力例

```
a+b=7
```

このプログラムの下線部について、関数を用いたものに書き換えたもの示す。

プログラム例１

```
#include <stdio.h>

int add(int x, int y) // 関数本体の定義
{
        int z;
        z = x + y;
        return z;
}

int main(void) // メイン関数
{
        int a = 3, b = 4, c;
        c = add(a, b);
        printf("a+b=%d¥n", c);
        return 0;
}
```

出力例

```
a+b=7
```

このプログラムの場合にどのような順番で実行されるかを下図に示す。

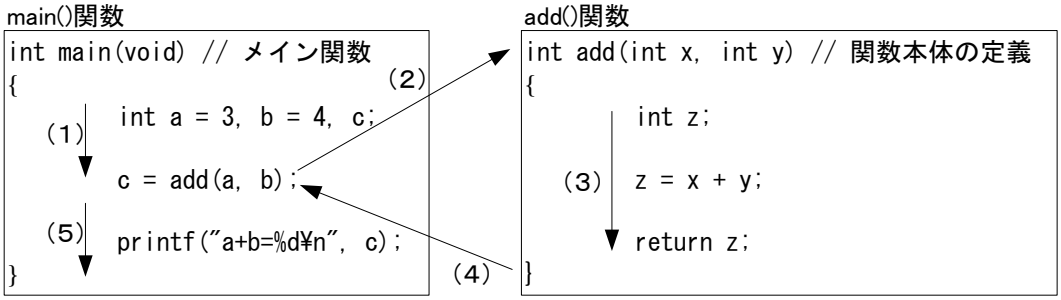

図１　プログラムの流れ

（１）プログラムの開始場所はいつでも main()関数から始まる。順番に実行し、add(a,b)が実行される。これは関数を呼び出すともいう。
（２）add()関数本体に実行が移る。この際、add()関数呼び出し時の a と b の値がそれぞれ add()関数本体の x と y にそれぞれコピーされる。
（３）add()関数本体が順番に実行される。
（４）main()関数に処理が戻る。その際、add()関数の z の値が main()関数の変数 c に代入される。
（５）main()関数の残りの処理が実行される。

ここで、add(a, b)や int add(int x, int y)で用いられている a、b、x、y は関数に与えるパラメータであり、これらを「引数」という。特に main()関数本体で add()関数呼び出しに使っている add(a, b)の a と b は実際に処理に使っている値であることから「実引数」といい、それに対して add( )関数本体の定義に用いている int x と int y は「仮引数」という。また add()関数本体終了時に return z;と書き、main()関数に z の値を戻していることから、z を「返却値」という。

プログラム例１でも動作するが、より一般的な書き方として、「プロトタイプ宣言」を用いる.

プロトタイプ宣言

プロトタイプ宣言を用いて，プログラム例１を修正すると，以下のようになる.

プログラム例２

```
#include <stdio.h>

int add(int x, int y); // プロトタイプ宣言

int main(void) // メイン関数
{
        int a = 3, b = 4, c;
        c = add(a, b);
        printf("a+b=%d¥n", c);
        return 0;
}

int add(int x, int y) // 関数本体の定義
{
        int z;
        z = x + y;
        return z;
}
```

出力例

```
a+b=7
```

このプロトタイプ宣言はコンパイラに関数名とその関数の引数に用いる変数の型と個数、及び返却値に用いる変数の型をプログラムの始めの部分で教える役割がある。これらがプロトタイプ宣言と関数本体で異なっている場合にはコンパイルエラーが発生する。また、プログラム例１では add()関数、main()関数の順に記述したが、これは main()関数、add()関数の順に記述するとコンパイルエラーとなる。これは宣言する前に利用したためである。

プロトタイプ宣言は以下の書式に従う。

書式
返却値の型　関数名（引数名１の型 引数名１, …, 引数名 n の型 引数名 n）;
使用例（プログラム例２の抜粋）
int add(int x, int y);

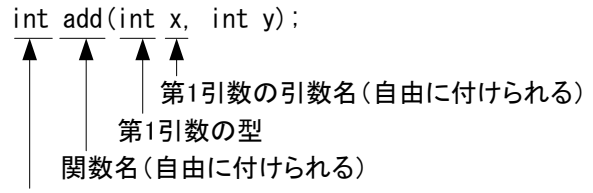

図２　関数（プロトタイプ宣言）の説明

関数では一般的に引数によって値を受け取り、return で返却値を返すが、中には引数が必要無い関数や返却値の無い関数もある。そのような場合、「引数が無い」ことや「返却値が無い」ことを明示するために以下のように「void 型」を使う。

```
void func(void) ;
```

もう少し、具体例を示していく。返却値が無い場合は返却値の型は以下のように void 型を用いる。

```
void func(int x, int y);
```

引数がない場合は以下のように void 型を用いる。

```
int func(void);
```

補足
返却値や引数を省略した場合には、返却値の型は int 型となり、引数は void 型となる。 例　　func(); 基本的には省略しないでしっかり書くこと。 int func(void);

関数本体の定義

関数本体の定義方法について述べる。これは以下の書式に従う。

書式
返却値の型　関数名（引数名１の型 引数名１, …, 引数名 n の型 引数名 n） { 　　変数定義 　　文１； 　　文２； 　　： 　　：

```
        return 返却値;
}
```

使用例（プログラム例１の抜粋）

```
int add(int x, int y) // 関数本体の定義
{
        int z;
        z = x + y;
        return z;
}
```

ここでは return 文について説明を加えていく。return 文は関数において、値を返す必要がある時、以下の書式で用いる。

書式

```
return 値; または、return 式;
```

また、値を返す必要のないときには「return;」とだけ記述することになるが、通常、関数の最後でリターンする場合にはこれを省略する。

使用例

```
void func(void)
{
        処理
        return;          ⇒       省略する
}
```

また、関数中で return 文を複数書く場合もある。例えば引数で渡された変数の値の正負を判定し、絶対値を返却値として返すプログラムの場合、以下のようなプログラムが考えられる。

プログラム例３

```
#include <stdio.h>

int my_abs(int x); // プロトタイプ宣言

int main(void) // メイン関数
{
        int a = -3;
        printf("%dの絶対値は%dです¥n", a, my_abs(a));
        return 0;
}

int my_abs(int x) // 関数本体の定義
{
        if(x>=0) return x;
        else     return -x;
}
```

出力例

```
-3 の絶対値は 3 です
```

練習

いくつかのプログラムを示す。プログラム中の下線部の処理について関数を用いたプログラムに変更した例を示していく。

プログラム例４

```
#include <stdio.h>

int main(void)
{
	printf("関数の練習¥n");
	return 0;
}
```

変更後

```
#include <stdio.h>

void func(void);

int main(void)
{
	func();
	return 0;
}

void func(void)
{
	printf("関数の練習¥n");
}
```

このプログラムの場合には出力処理を関数で記述することになるが、引数と返却値は必要ないため、プロトタイプ宣言は「void func(void);」のようになる。

プログラム例５

```
#include <stdio.h>

int main(void)
{
	int n=10;
	printf("%d¥n", n);
	return 0;
}
```

変更後

```
#include <stdio.h>

void func(int s);

int main(void)
{
	int n=10;
	func(n);
	return 0;
}

void func(int s)
{
```

```
        printf("%d¥n", s);
}
```

このプログラムの場合も出力処理を関数で記述することになるが、変数 n を出力に使っているため、この値を引数として関数に渡す必要がある。返却値は必要ないため、プロトタイプ宣言は「void func(int s);」のようになる。

プログラム例６

```
#include <stdio.h>

int main(void)
{
        int a=5, b=1, c;
        c = a*b;
        printf("c = %d¥n", c);
        return 0;
}
```

変更後

```
#include <stdio.h>

int mul(int x, int y);

int main( void )
{
        int a=5, b=1, c;
        c = mul(a, b);
        printf("c = %d¥n", c);
        return 0;
}

int mul(int x, int y)
{
        int z;
        z = x*y;
        return z;
}
```

このプログラムの場合では 2 つの変数を用いて、乗算を行いその結果を出力する。出力には変数を用いているので、その処理で得られる値が必要となる。この場合関数の引数は整数型で 2 つ、返却値として整数型が必要となる。これは main 関数での変数宣言の型に合わせる必要がある。よって、プロトタイプ宣言は「int mul(int x, int y);」のようになる。

プログラム例７

```
#include <stdio.h>

int main(void)
{
        int a, b, c;

        printf("二つの整数値を入力¥n");
        scanf("%d %d", &a, &b);
```

```
        if(a<=b){
                c=a;
        }
        else{
                c=b;
        }

        printf("小さい方 = %d¥n", c);

        return 0;
}
```

変更後

```
#include <stdio.h>

int min(int x, int y);

int main(void)
{
        int a, b, c;

        printf("二つの整数値を入力¥n");
        scanf("%d %d", &a, &b);

        c = min(a, b);

        printf("小さい方 = %d¥n", c);

        return 0;
}

int min(int x, int y)
{
        if(x<=y) return x;
        else     return y;
}
```

このプログラムの場合はプログラム例６の場合と同様である。よって、プロトタイプ宣言は「int min(int x, int y)」のようになる。

演習

プログラム例１～７を作成して、出力を確認しなさい。