[プログラミング入門Ｉ　（４）］ 関数とは

プログラミング言語のもっとも重要な要素の１つが関数です。数学での関数は、パラメータがあり、それを何らかの値にマッピングするというものでした。たとえば、f(x) = 2x+1 とかというやつです。これは、x をパラメータとしてｘに 2 とかけて１をたした値とするという関数です。もちろん、プログラミング言語での関数も、これと同じことができます。たとえば、f(x)は以下のようにかくことができます。関数の値は、return 文を使います。呼び出しは、以下のように書きます。以下のプログラムでは、f(10)の値を計算しています。

```
int f(int x){
   return x*2 + 1;
}
main(){
   int x;
   x = f(10);
   printf("x = %d¥n",x);
}
```

プログラミング言語での関数は数学での関数のような役割以外にもっと重要な役割をもっています。それは、**機能を一まとめにしておくという役割**です。C では「**関数**」（**function**）と呼んでいますが、そのため、他の言語では関数以外にも「**手続き**」**(procedure)**と呼んでいる場合もあります。たとえば、printf や scanf も関数です。入力や出力したりという機能を関数（手続き）としてまとめておけば、それをいちいち書かなくても、それを「呼び出す」ことによって、その機能を使うことができるようになります。関数がなければ、同じような操作を繰り返して書いておかなくてはなりません。これでは、大きいプログラムは作ることができません。
実際、プログラムは関数の集まりとして作られています。C の main()も関数です。いろいろな動作をするプログラムはこの機能を持つ関数の集まりとして作ることによって、実現されています。

関数定義と関数呼び出し

関数の定義は以下の形式で書きます。

関数の返す値のデータ型　関数名（パラメータのデータ型　パラメータ名、....）
{　/* あれば、変数の宣言 */
...関数のプログラム...
return 関数から返す値の式；
}

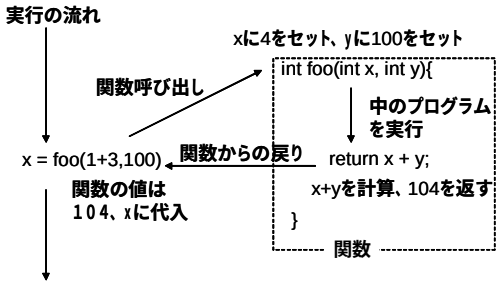
return から始まる **return 文**は、最後でなくてもかまいません。return 文が実行された時点で、関数の実行は終了し、return 後に書かれた式の値が呼び出し側に値が返されます。関数名の規則は変数と同じで英数字と_だけが使えます。「関数の返す値のデータ型」は省略できますが、その場合関数の返す方は int（整数）と解釈されます。例えば、今まで書いてきた main はその一つです。しかし、なるべくつけるようにしましょう。
関数呼び出しは、

関数名（引数の式、....）

と書きます。これは関数の返す値のデータ型を持つ式として使えます。例えば、

x = foo(1,y)+1;

というような式の中で使えます。関数にパラメータとして与える値のことを「引数」（ひきすうとよむ）といいます。関数呼び出しが行われると、引数の式の値が対応するパラメータの変数として参照することができます。図に関数の実行の流れを示します。関数の実行が終わると、実行の流れは関数呼び出しのところに戻り、関数呼び出しの式の値として、関数の返した値が参照されます。



関数の呼び出しをする場合は、その前に関数の型の宣言が必要です。これは、関数の宣言の本体（{ }以外の部分）を除いたものです。

関数の返す値のデータ型　関数名（パラメータのデータ型　パラメータ名、....）；

最後の；をわすれないでください。これを関数の**プロトタイプ宣言**といいます。これは関数の返す値とそのパラメータのデータ型をあらかじめ宣言しておくものです。関数定義の外に、通常プログラムのはじめに書いておきます。返す値のデータ型が整数の場合には省略できますが、これもなるべく行うようにしてください。ちなみに、#include で stdio.h などを読み込んでいるのは、ライブラリ関数のプロトタイプ宣言などを読み込んでいます。また、sin や cos などのライブラリを使う場合には、このライブラリ関数のプロトタイプ宣言をしてある、math.h を include する必要があります。
関数の書き方についてまとめておきましょう。

- まず、先頭にプロトタイプ宣言をしておく。

- 関数の呼び出しはその後。
- プロトタイプ宣言をしておけば、関数の本体を定義はどこでもかまいません。

下の例に、２つの数の大きいほうを返す関数を使う例を示します。関数がなぜ必要なのかを例で考えてみることにしましょう。関数はある機能を一まとめにしたものです。例えば、４つの数の最大値を求めたい場合、if 文をたくさん書くよりも、

```
imax(imax(1,3),imax(5,4))
```

と書いたほうが、簡単にすみます。

```
#include <stdio.h>
int imax(int a,int b);
main(){
 int x,y,z;
 scanf(“%d”,&x);
scanf(“%d”,&y);
 z = imax(x,y);
 printf(“max is %d¥n”,z);
}
int imax(int a, int b){
 if(a > b) return a;
 else return b;
}
```

手続きとしての関数

関数は機能を一まとめにしておくものですので、値を返さない場合もあります。その場合は、関数の返すデータ型に void と書きます。これは、値を返さないものという意味です。

```
void   関数名（パラメータのデータ型   パラメータ名、....）
{     /* あれば、変数の宣言 */
        ...関数のプログラム...
        return；
}
```

この場合は return には式をつけません。また、関数は最後まで実行すると自動的に return しますので、最後で return する場合には書かなくてもかまいません。この場合、値が返されないので、関数の呼び出しは式の中に書かずに、文として

```
関数（引数、....）；
```

と書きます。

いろいろな変数：局所変数、大域変数、静的変数の使い方

これまで、変数はすべて関数の中に宣言してきました。関数の中で宣言されている変数を**局所変数**（local variable, 自動変数:automatic variable ともいう）といいます。**局所変数**（local variable）は、関数の中で一時的に使うデータを格納しておくのに使います。それに対し、関数の外で宣言された**大域変数**(global variable)はプログラム全体で使うデータを格納しておくために使います。

関数は数学の関数のように、パラメータのみを使って、値を計算するだけではありません。通常、関数はある機能ごとに作り、プログラムをわかりやすく部分部分に分けて書くために使います。大きなプログラムを作る場合にはこのようにして部分に分けてかくことは非常に重要です。例えば、カウンタがあり、それを増やしたり（increment）減らしたり(decrement)という関数を次のようにしてつくっておきます。

```
int counter;
void increment() { counter += 1; }
void decrement() { counter -= 1; }
```

これは非常に簡単な例ですが、このようにしておいて、この関数を使っておけば、counter に対する演算をいろいろなところにかくよりも、関数の名前、つまりやりたいことの名前でかくことができるようになり、プログラムがわかりやすくなります。この関数は値をかえしませんが (void をつかった手続き)、**手続きに名前をつけて使うこと**ができるようになります。

多くの局所変数は、大域変数でもよい場合があります。一時的に大域変数をつかってもよい場合です。例えば、int t；と大域変数を宣言しておけば、

```
int foo(x,y) {  t = x + y; return t; }
```

でもかまいません。しかし、大域変数は他の関数でも使われる可能性があるので、

```
int goo(x,y) { t = x-y; z = foo(x,y); .... ; return t; }
```

という場合には、関数 goo でセットした t の値が関数 foo で壊される（これが**副作用**!）ので、思いがけないエラーがおきてしまうことになります。これが大きなプログラムになると間違いを見つけるのが大変になってしまいます。**局所的しか使わない一時的なデータにはできるだけ局所変数を使い、必要なときだけ大域変数を使うようにしましょう。**

関数が終わっても値を保持したい場合につかう**静的変数**は他の関数には使われたくないが、値を保持したい場合につかいます。（静的変数とは static を宣言につけた変数）