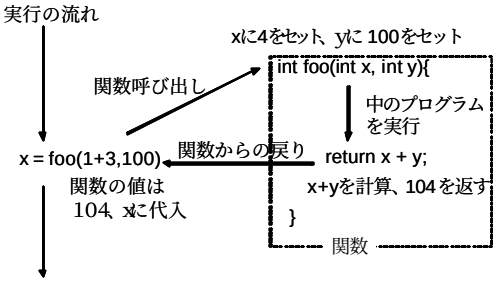
[プログラミング序論 II　3 回目　２００２・９・２６]

関数定義と関数呼び出し

プログラミング言語での関数の役割とは、ある機能をひとまとまとまりにしておくことです。そのため、他の言語では、手続き(procedure)とかサブルーチン(subroutine)と呼ばれることもあります。
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（教科書には文字型と書いていますが、これは正しくありません)。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(教科書 181 ページ)。
なお、教科書では関数のパラメータの名前とパラメータのデータ型の宣言を別にしてありますが、最近のCの仕様では一緒に書いておくほうが主流です。（もちろん、別に書いてもよい）

```
#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;
}
```

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。

右の例に、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。組み合わせの数（２項係数）を計算する場合には、n 個から r 個選ぶ組み合わせ ${}_nC_r = n!/(r!(n-r)!)$で与えられます（教科書 172 ページ)。n!、つまり n の階乗を計算する関数を定義しておけば、これを利用すれば、例のようにコンパクトに書くことができます。

```
int combination(int n,int r);
int factorial(int n);
main()
{
    printf("c = %d¥",combination(10,5));
}
int combination(int n,int r)
{
 return factorial(n)/(factorial(r)*factorial(n-r));
}
int factorial(int n)
{   int i,k;
    k = 1;
   for(i=1; i<=n; i++) k *= i;
    return k;
}
```

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

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。パラメータとして宣言されている変数も局所変数です。はじめに引数の値がセットされて以外は局所変数と同じです。局所変数は宣言されている関数の中でしか使うことができません。これに対して、関数の外で宣言された変数を**大域変数** (global variable)といいます。このような変数は、関数の間で共通に使うデータのための変数を宣言するために使います。関数のプロトタイプ宣言と同じように、大域変数は変数を使う前に（できれば、プログラムの先頭で宣言しなくてはなりません。void で宣言した手続きとしての関数は、大域変数で宣言されたデータに対してなんらかの操作を行うために使います。このように大域変数の値を変更することを**関数の副作用**といいます。

局所変数は、その宣言された関数が終了すると値が無効になってしまいます。大域変数は関数が終わっても値は保持されますので、もしも、関数が終わってもなにか値を保持する必要がある場合には大域変数にいれておくことができます。また、関数から返すことのできる値は１つなので、たくさんの値を返したい場合には、大域変数を用意してそこにいれて、関数からも戻ったときにその変数を参照することでできます。関数が終わっても値を保持したい場合には、**静的変数**を使うことができます。これは局所変数の宣言の先頭に static というキーワードをつけたものです。変数を参照できる範囲のことを変数の**参照範囲**（scope）といいます。局所変数の有効範囲は関数の中、大域変数の有効範囲は変数が宣言され以降のプログラムです。変数の値が保持される期間を**有効期間**といいます。局所変数の有効期間は関数の実行中、大域変数の有効期間はプログラム実行中です。静的変数とは参照範囲が関数内で、有効期間はプログラム実行中という変数です。

配列の引数

式で計算した値だけでなく、配列を関数に引数として渡すためには、次のように書きます。

```
関数の返す値のデータ型　関数名（配列パラメータのデータ型　配列のパラメータ名[]、....）
{  ...後は同じ...  }
```

関数宣言でパラメータ宣言のところにでは、サイズなしで[]をつけておきます。関数の中では、パラメータで宣言された配列名は配列と同じように A[i]というように配列と同じように使うことができます。呼び出し側では、次のように配列名を引数にします。

```
関数名（引数の配列名、....）
```

文字列も文字の配列ですから、同じように関数に引き渡すことができます。配列については後で説明するポインタと深く関連しています。また、２次元以上の配列の渡し方についても、後で時間があれば説明します。

- 配列の入っているデータの合計を求めるプログラム（教科書 184 ページ）

小テスト問題

最大１０文字の文字列を入力し、逆順に出力するプログラムを書きなさい。例えば、apple は、elppa と出力する。文字列とは C では、文字コード 0 で終わる文字の配列であることに注意すること。

次回は、関数の再帰呼び出しについて