[練習 1] 2 つの整数 a, b を受けとり、a^b を求める関数 `power(a, b)` を作りなさい。そして、この関数を呼ぶことで $i^j + j^i$ を計算するプログラムを書きなさい。

値を返さない関数

値を返さない関数というのもあります。この時は、

```
void 関数名 (引数)
```

のように書きます。そして、値を返さないので、その関数から抜ける時は

```
return ;
```

とだけ書きます。

[練習 2] 正整数 m を引数にとり、* を m 個横に出力する関数 prstar を作りなさい。

次のように、引数をとらず値も返さない関数も書けます。

```
void printmsg(void)
{
    printf("Hello, guys!\n");
}
```

以上のように、C 言語における関数は「ある値を受けとって、計算した値を返す」という普通の関数の意味より広く、なんらかの働きをする部品と考えてよいでしょう。すなわち、処理を機能ごとに分割して、それぞれの機能を関数にするのがよいプログラムといえます。

main という名の関数

実は、必ず最初に呼ばれることを除いて、main も一つの関数に過ぎません。main は普通、型宣言を省略して書きます。関数は返り値の型宣言を省略すると int とみなされるので、厳密に書けば、

```
int main(void)
```

と同じです。

局所変数と大域変数

今まで、変数は main やユーザー関数の中でのみ宣言してきました。関数内で宣言された変数は、その関数の中でだけ参照・代入することができます。このような変数を**局所変数**といいます[1]。しかしながら場合によっては、いくつかの関数で変数を共有して使いたいことがあります。このような時は、関数の宣言と同じレベルで変数を宣言します。これを**大域変数** (または外部変数) といいます。大域変数をうまく使うと、効率の良いプログラムを書くことが出来る一方、**使い過ぎるとプログラム全体が見にくくなるので、充分注意が必要です**[2]。

[練習 3] 次のプログラムを実行し、変数 i が局所的であることを確認しよう。

```
 1  #include <stdio.h>
 2
 3  void foo(void);                          /* プロトタイプ宣言 */
 4
 5  main()
 6  {
 7    int i;
 8
 9    for(i=0; i<3; i++) {
10       printf("main...i = %d\n", i);
11       foo();                              /* 引数がないときでも括弧は書く */
12    }
13  }
14
15  void foo(void)
16  {
17    int i;
18
19    for(i=0; i<3; i++) {
20       printf("   foo...i = %d\n", i);
21    }
22  }
```

[1]関数の中を更に細かく区切って局所変数を作ることも出来ますが、さしあたり必要ないでしょう。
[2]特に複数人で共同開発する時は、他の人のプログラムに影響を与える可能性があるので無闇に使えません。

変数名の有効範囲

関数の中で変数が使われると、まず局所変数が探され、もしなければ大域変数が探されます。次の例では何が出力されるでしょうか？

```
 1  #include <stdio.h>
 2  void foo(void);                  /* プロトタイプ宣言 */
 3  int x = 100;                     /* 大域変数 x を初期値 100 として宣言 */
 4
 5  main () {
 6    int x = 200;                   /* 局所変数 x を初期値 200 として宣言 */
 7    foo();                         /* ユーザ関数 foo を呼び出す */
 8  }
 9
10  void foo(void) {                 /* ユーザ関数 foo の定義 */
11    printf("x = %d\n", x);         /* x をプリント */
12  }
```

この場合、関数 foo の中に x という局所変数がないので、foo の中に出てきた x は大域変数だとみなされます。つまり、100 が表示されます。

[練習 4] 次のプログラムでは何が表示されるか考えなさい。

```
 1  #include <stdio.h>
 2  int add5(int);
 3  void add0(int);
 4  int  k=100;
 5
 6  main() {
 7     int i=10, j;
 8     j=add5(i);
 9     printf("(main) i=%d, j=%d, k=%d\n", i,j,k);
10  }
11
12  int add5(int n) {
13     int i=20;
14     add0(n);
15     printf("(add5) i=%d, k=%d, n=%d\n", i,k,n);
16     return(n+5);
17  }
18
19  void add0(int k) {
20     printf("(add0) k=%d\n",k);
21     return;
22  }
```