

# 第七章 用函数实现模块化程序设计

作者：石璞东

参考资料：《C程序设计（第四版）》谭浩强

## 7.1 为什么要用函数

函数就是功能，每一个函数用来实现一个特定的功能，函数的名字反应其代表的功能。

在设计一个较大的程序时，往往把它分为若干个程序模块，每一个模块包含一个或多个函数，每个函数实现一个特定的功能。一个C程序可由一个主函数和若干个其他函数构成。由主函数调用其他函数，其他函数也可以互相调用，同一个函数可以被一个或多个函数调用任意多次，如图所示。

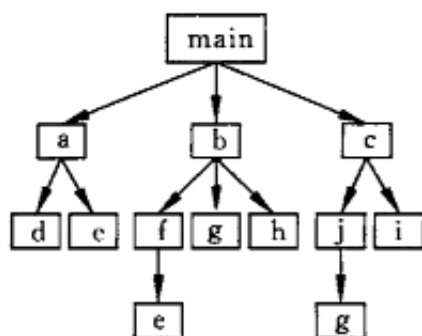


图 7.1

### 例7.1 输出特定形状

```
1  #include<stdio.h>
2  int main(){
3      int print_star();
4      int print_message();
5      print_star();
6      print_message();
7      print_star();
8      return 0;
9  }
10 int print_star(){
11     printf("*****\n");
12     return 0;
13 }
14
15 int print_message(){
16     printf("HelloWorld!\n");
17     return 0;
```

运行效果如下所示：

```

7
8 #include<stdio.h>
9 int main(){
10     int print_star();
11     int print_message();
12     print_star();
13     print_message();
14     print_star();
15     return 0;
16 }
17
18 int print_star(){
19     printf("*****\n");
20     return 0;
21 }
22 int print_message(){
23     printf("HelloWorld!\n");
24     return 0;
25 }
26
27
28

```

```

*****
HelloWorld!
*****
Program ended with exit code: 0

```

在程序中，定义 `print_star` 函数和 `print_message` 函数的位置都是在 `main` 函数的后面，在这种情况下，应当在 `main` 函数之前或 `main` 函数中的开头部分，对以上两个函数进行声明。函数声明的作用是把有关函数的信息（函数名、函数类型、函数参数的个数与类型）通知编译系统，以便在编译系统对程序进行编译时，在进行到 `main` 函数调用 `print_star()` 和 `print_message()` 时知道它们是函数而不是变量或其他对象，此外，还对调用函数的正确性进行检查，如类型、函数名、参数个数、参数类型等是否正确。

【注】：

- 一个 `C` 程序由一个或多个程序模块组成，每一个程序模块作为一个源程序文件，对较大的程序，一般不希望把所有内容全部放在一个文件中，而是将它们分别放在若干个源文件中，由若干个源程序文件组成一个 `C` 程序，这样便于分别编写和编译，提高调试效率。一个源程序文件可以为多个 `C` 程序共用。
- 一个源程序文件由一个或多个函数以及其他有关内容（如指令、数据声明与定义等）组成。一个源程序文件是一个编译单位，在程序编译时是以源程序文件为单位进行编译的，而不是以函数为单位进行编译的。
- `C` 程序的执行是从 `main` 函数开始的，如果在 `main` 函数中调用其他函数，再调用后流程返回到 `main` 函数，在 `main` 函数中结束整个程序的运行。
- 所有函数都是平行的，即在定义函数时是分别进行的，是互相独立的。一个函数并不从属于另一个函数，即函数不能嵌套定义。函数间可以互相调用，但不能调用 `main` 函数，`main` 函数是被操作系统调用的。
- 从用户使用的角度看，函数有两种。

- 库函数：由系统提供，用户不必自己定义，可直接使用，根据不同的 c 编译系统，提供的库函数的数量和功能会有一些不同；
- 用户自己定义的函数：用以解决用户专门需要的函数；
- 从函数的形式看，函数分两类。
  - 无参函数：主调函数无需向被调用函数传递数据，一般仅用来执行指定的一组操作，其函数值可以带回或不带回，一般以不带回函数值的居多；
  - 有参函数：主调函数向被调用函数传递数据，并得到返回值供主调函数使用；

## 7.2 怎样定义函数

### 7.2.1 为什么要定义函数

定义函数应该包括以下几个内容：

- 指定函数的名字，以便以后按名调用；
- 指定函数的类型，即函数返回值的类型；
- 指定函数的参数的名字和类型，以便在调用函数时向它们传递数据，对无参函数不需要这项；
- 指定函数应当完成什么操作，也就是函数是做什么的，即函数的功能。

对于 c 编译系统提供的库函数，只需通过 `#include` 命令引入头文件即可。

### 7.2.2 定义函数的方法

- 定义无参函数

```
1  类型名 函数名(){  
2      //函数体  
3  }  
4  或  
5  类型名 函数名(void){  
6      //函数体  
7  }  
8  其中，函数名后面括号内的void表示“空”，即函数没有参数。
```

- 定义有参函数

```
1  类型名 函数名(形式参数列表){  
2      //函数体  
3  }
```

- 定义空函数

```
1 | 类型名 函数名 ( ) { }
```

## 7.3 调用函数

### 7.3.1 函数调用的形式

函数调用的一般形式为：

```
1 | 函数名 ( 实参表列 );
```

【注】：若实参表列中包含多个实参，则各参数见用逗号隔开。

3种函数调用方式：

- 函数调用语句

把函数调用单独作为一个语句，这时不要求函数带返回值，只要求函数完成一定的操作。

- 函数表达式

函数调用出现在另一个表达式中，如 `c = max(a,b);`，`max(a,b)` 是一次函数调用，它是赋值表达式中的一部分，此时要求函数带回一个确定的值以参加表达式的计算。

- 函数参数

函数调用作为另一个函数调用时的实参，如 `m = max(a,max(b,c))`，其中 `max(b,c)` 是一次函数调用，它的值作为 `max` 另一次调用的实参。

### 7.3.2 函数调用时的数据传递

- 形式参数：定义函数时函数名后面括号中的变量称为**形式参数**；
- 实际参数：在主调函数中调用一个函数时，函数名后面括号中的参数称为**实际参数**；

【注】

- 实参可以是常量、变量或表达式；
- 实参与形参的类型应相同或赋值兼容。

### 7.3.3 函数调用的过程

- 在定义函数中指定的形参，当未出现函数调用时，它们并不占内存中的存储单元，在发生函数调用时，其形参才会被临时分配内存单元；
- 函数调用结束时，形参单元会被释放，但实参单元仍保留并维持原值，没有改变，这是因为实参与形参是两个不同的存储单元；
- 实参向形参的数据传递是**值传递**，单向传递，只能由实参传递给形参，而不能由形参传给实参，实参和形参在内存中占有不同的存储单元，实参无法得到形参的值；

## 7.3.4 函数的返回值

- 函数的返回值是通过函数中的 `return` 语句获得的，一个函数中可以有一个以上的 `return` 语句；
- 应当在定义函数时指定函数值的类型；
- 在定义函数时指定的函数类型一般应该和 `return` 语句中的表达式类型一致，若不一致，则以函数类型为准；

### 例7.2 函数类型决定返回值类型

```
1  #include<stdio.h>
2  int main(){
3      int max(float x,float y);
4      float a,b;
5      int c;
6      printf("请输入两个数字: ");
7      scanf("%f %f",&a,&b);
8      c = max(a,b);
9      printf("max = %d\n",c);
10     return 0;
11 }
12 int max(float x,float y){
13     float z;
14     z = x>y?x:y;
15     return z;
16 }
```

运行效果如下所示：

```

8  #include<stdio.h>
9  int main(){
10     int max(float x,float y);
11     float a,b;
12     int c;
13     printf("请输入两个数字: ");
14     scanf("%f %f",&a,&b);
15     c = max(a,b);
16     printf("max = %d\n",c);
17     return 0;
18 }
19 int max(float x,float y){
20     float z;
21     z = x>y?x:y;
22     return z;
23 }
24
25
26
27
28

```

```

请输入两个数字: 1.24 4.53
max = 4
Program ended with exit code: 0

```

## 7.4 对被调用函数的声明和函数原型

函数的首行称为函数原型，使用函数原型作声明是 C 的一个重要特点，用函数原型来声明函数能减少编写程序时可能出现的错误。

实际上，在函数声明中的形参名可以省写，而只写形参的类型，编译系统只关心和检查参数个数和参数类型，而不检查参数名，因为在调用函数时只要求保证实参类型与形参类型一致，而不必考虑形参名是什么。

一般情况下，函数原型的一般形式有两种：

- 函数类型 函数名 (参数类型1 参数名1, 参数类型2 参数名2, ...参数类型n 参数名n)
- 函数类型 函数名 (参数类型1, 参数类型2, ..., 参数类型n)

**例7.3** 输入两个实数，用一个函数求出它们之和。

```

1  #include<stdio.h>
2  int main(){
3     float add(float x,float y);
4     float a,b,sum;
5     printf("请输入两个数字: ");
6     scanf("%f %f",&a,&b);
7     sum = add(a,b);

```

```

8     printf("sum = %.2f\n",sum);
9     return 0;
10 }
11 float add(float x,float y){
12     float z;
13     z = x+y;
14     return z;
15 }

```

运行结果如下所示：

```

/
8  #include<stdio.h>
9  int main(){
10     float add(float x,float y);
11     float a,b,sum;
12     printf("请输入两个数字: ");
13     scanf("%f %f",&a,&b);
14     sum = add(a,b);
15     printf("sum = %.2f\n",sum);
16     return 0;
17 }
18 float add(float x,float y){
19     float z;
20     z = x+y;
21     return z;
22 }
23
24
25
26
27

```



```

请输入两个数字: 1 4
sum = 5.00
Program ended with exit code: 0

```

如果已在文件的开头（在所有函数之前），已对本文件中所调用的函数进行了声明，则在各函数中不必对其所调用的函数再作声明。

```

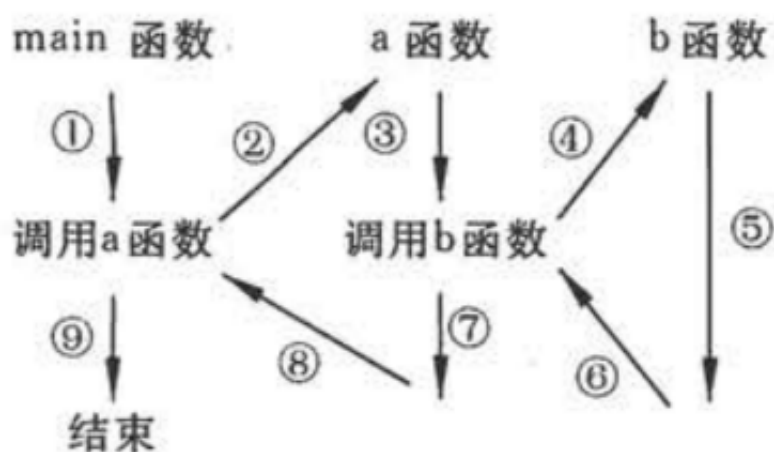
char letter(char, char);           //以下 3 行在所有函数之前,且在函数外部
float f(float, float);
int i (float, float);

int main()                         //在 main 函数中要调用 letter, f 和 i 函数
{
    :
}
//下面定义被 main 函数调用的 3 个函数
char letter(char c1, char c2)     //定义 letter 函数
{
    :
}
float f(float x, float y)         //定义 f 函数
{
    :
}
int i(float j, float k)           //定义 i 函数
{
    :
}

```

## 7.5 函数的嵌套调用

c 语言的函数定义是互相平行、独立的，也就是说，在定义函数时，一个函数内是不能再定义另一个函数，也就是不能嵌套定义，但可以嵌套调用函数。



- 执行 main 函数的开头部分；
- 遇函数调用语句，调用函数 a，流程转去 a 函数；
- 执行 a 函数的开头部分；
- 遇函数调用语句，调用函数 b，流程转去 b 函数；
- 执行 b 函数，如果再无其他嵌套的函数，则完成 b 函数的全部操作；
- 返回到 a 函数中调用 b 函数的位置；
- 继续执行 a 函数中尚未执行的部分，直到 a 函数结束；



- 返回 `main` 函数中调用 `a` 函数的位置；
- 继续执行 `main` 函数的剩余部分直到结束；

**例7.5** 输入4个整数，找出其中最大的数，用函数的嵌套来处理。

```
1  #include<stdio.h>
2  int main(){
3      int max4(int a,int b,int c,int d);
4      int a,b,c,d,max;
5      printf("请输入四个数字: ");
6      scanf("%d %d %d %d",&a,&b,&c,&d);
7      max = max4(a, b, c, d);
8      printf("最大值为%d\n",max);
9      return 0;
10 }
11 int max4(int a,int b,int c,int d){
12     int max2(int a,int b);
13     return max2(max2(max2(a, b),c ),d);
14 }
15 int max2(int a,int b){
16     return a>b?a:b;
17 }
```

运行结果如下所示：

```

8  #include<stdio.h>
9  int main(){
10     int max4(int a,int b,int c,int d);
11     int a,b,c,d,max;
12     printf("请输入四个数字: ");
13     scanf("%d %d %d %d",&a,&b,&c,&d);
14     max = max4(a, b, c, d);
15     printf("最大值为%d\n",max);
16     return 0;
17 }
18 int max4(int a,int b,int c,int d){
19     int max2(int a,int b);
20     return max2(max2(max2(a, b),c ),d);
21 }
22 int max2(int a,int b){
23     return a>b?a:b;
24 }
25
26
27 |
28
29

```

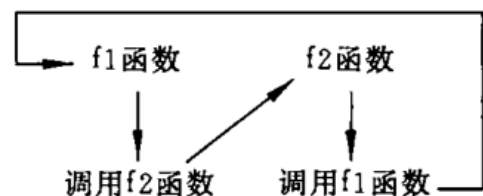
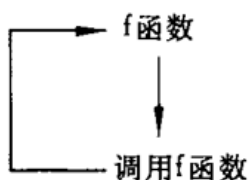
请输入四个数字: 1 2 3 4  
 最大值为4  
 Program ended with exit code: 0

## 7.6 函数的递归调用

一个递归问题可以分为两个阶段：回溯和递推。

在调用函数 `f` 的过程中，又要调用 `f` 函数，这是直接调用本函数。

如果在调用 `f1` 函数过程中要调用 `f2` 函数，而在调用 `f2` 函数过程中又要调用 `f1` 函数，就是间接调用本函数。



### 例7.5 年龄问题

```

1  #include<stdio.h>
2  int age(int n){
3      int c;
4      if(n==1){

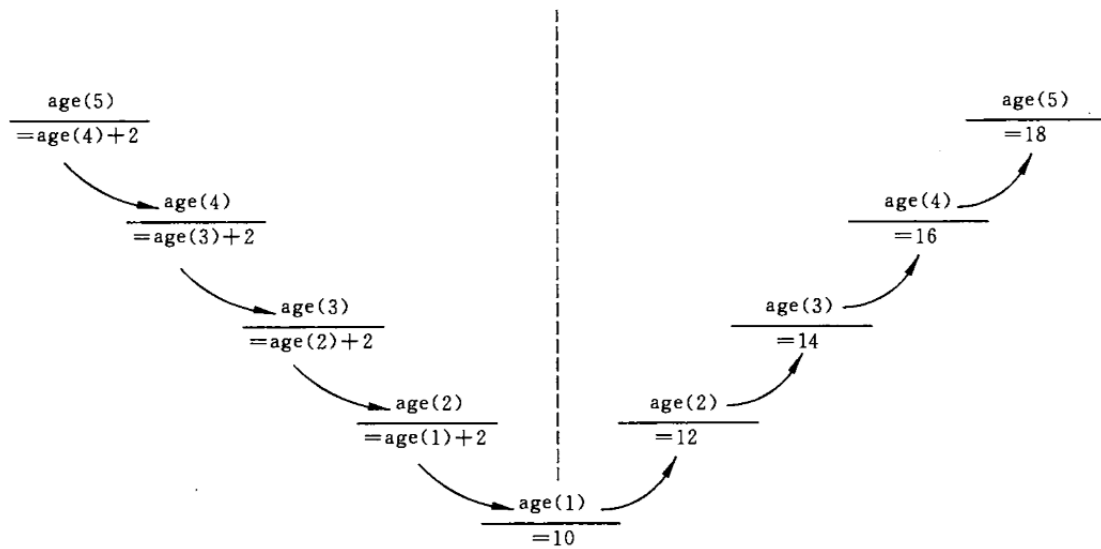
```

```

5      c = 10;
6  }else{
7      c = age(n-1)+2;
8  }
9  return c;
10 }
11 int main(){
12     int age(int n);
13     printf("第五个学生的年龄为: %d\n",age(5));
14     return 0;
15 }

```

回溯与递推示意图：



运行结果如下所示：

```

8  #include<stdio.h>
9  int age(int n){
10     int c;
11     if(n==1){
12         c = 10;
13     }else{
14         c = age(n-1)+2;
15     }
16     return c;
17 }
18 int main(){
19     int age(int n);
20     printf("第五个学生的年龄为: %d\n", age(5));
21     return 0;
22 }
23
24
25

```

第五个学生的年龄为: 18  
Program ended with exit code: 0

## 例7.6 求阶乘

```

1  #include<stdio.h>
2  int main(){
3      int fac(int n);
4      int n;
5      int res;
6      printf("请输入一个数字: ");
7      scanf("%d",&n);
8      res = fac(n);
9      printf("%d的阶乘值为%d\n",n,res);
10     return 0;
11 }
12
13 int fac(int n){
14     int f;
15     if(n < 0){
16         printf("当前值不合法!\n");
17     }else if(n==0 || n==1){
18         f = 1;
19     }else{
20         f = fac(n-1)*n;
21     }
22     return f;
23 }

```

```

8  #include<stdio.h>
9  int main(){
10     int fac(int n);
11     int n;
12     int res;
13     printf("请输入一个数字: ");
14     scanf("%d",&n);
15     res = fac(n);
16     printf("%d的阶乘值为%d\n",n,res);
17     return 0;
18 }
19
20 int fac(int n){
21     int f;
22     if(n < 0){
23         printf("当前值不合法!\n");
24     }else if(n==0 || n==1){
25         f = 1;
26     }else{
27         f = fac(n-1)*n;
28     }
29     return f;
30 }
31
32

```



请输入一个数字: 5

5的阶乘值为120

Program ended with exit code: 0

### 例7.7 汉诺塔问题

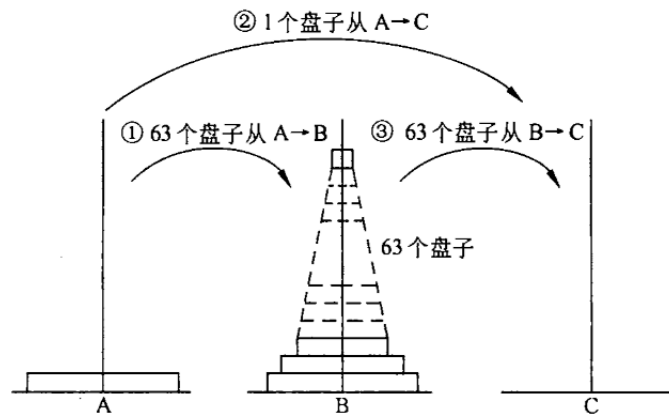
将  $n$  个盘子从 A 座移到 C 座的解题思路:

- 将 A 上  $n-1$  个盘借助 C 座先移到 B 座上;
- 把 A 座上剩下的一个盘移到 C 座上;
- 把  $n-1$  个盘从 B 座借助于 A 座移到 C 座上;

需要找到一个解决问题的思路,把看似复杂的问题简单化,使问题得以迎刃而解。老和尚会这样想:假如有另外一个和尚能有办法将上面 63 个盘子从一个座移到另一座。那么,问题就解决了。此时老和尚只须这样做:

- (1) 命令第 2 个和尚将 63 个盘子从 A 座移到 B 座;
- (2) 自己将 1 个盘子(最底下的、最大的盘子)从 A 座移到 C 座;
- (3) 再命令第 2 个和尚将 63 个盘子从 B 座移到 C 座。

见图 7.12。



```
1  #include<stdio.h>
2  int main(){
3      int hanoi(int n,char one,char two,char three);
4      int num;
5      printf("请输入盘子数量: ");
6      scanf("%d",&num);
7      printf("移动%d个盘子所需要的步骤为: \n",num);
8      hanoi(num, 'A', 'B', 'C');
9      return 0;
10 }
11 int hanoi(int n,char one,char two,char three){
12     void move(char x,char y);
13     if(n==1){
14         move(one,three);
15     }else{
16         hanoi(n-1, one, three, two);
17         move(one,three);
18         hanoi(n-1, two, one, three);
19     }
20     return 0;
21 }
22 void move(char x,char y){
23     printf("%c->%c\n",x,y);
24 }
```

```

8  #include<stdio.h>
9  int main(){
10     int hanoi(int n,char one,char two,char three);
11     int num;
12     printf("请输入盘子数量: ");
13     scanf("%d",&num);
14     printf("移动%d个盘子所需要的步骤为: \n",num);
15     hanoi(num, 'A', 'B', 'C');
16     return 0;
17 }
18 int hanoi(int n,char one,char two,char three){
19     void move(char x,char y);
20     if(n==1){
21         move(one,three);
22     }else{
23         hanoi(n-1, one, three, two);
24         move(one,three);
25         hanoi(n-1, two, one, three);
26     }
27     return 0;
28 }
29 void move(char x,char y){
30     printf("%c->%c\n",x,y);
31 }

```

```

请输入盘子数量: 3
移动3个盘子所需要的步骤为:
A->C
A->B
C->B
A->C
B->A
B->C
A->C
Program ended with exit code: 0

```

汉诺塔问题参考链接: [https://www.bilibili.com/video/BV1gb4y197TX/?spm\\_id\\_from=trigger\\_reload](https://www.bilibili.com/video/BV1gb4y197TX/?spm_id_from=trigger_reload)

## 7.7 数组作为参数

### 7.7.1 数组元素作函数实参

数组元素可以用作函数实参，不能用做形参，因为形参是在函数被调用时临时分配存储单元的，不可能为一个数组元素单独分配单元，因为数组是一个整体，在内存中占连续的一段存储单元。

当用数组元素做函数实参时，把实参的值传给形参，是值传递方式，数据传递的方向是从实参传到形参，单向传递。

例7.8 输入10个数，要求输出其中值最大的元素和该数是第几个数。

```

1  #include<stdio.h>
2  int main(){
3      int max(int x,int y);
4      int arr[10],m,n,i;
5      printf("请输入10个数字: ");
6      for (int i = 0; i < 10; i++) {
7          scanf("%d",&arr[i]);
8      }
9      for (i = 1,m = arr[0],n = 0; i < 10; i++) {
10         if(max(m,arr[i])>m){
11             m = max(m, arr[i]);
12             n = i;
13         }
14     }
15     printf("10个数字中最大者为: %d,其下标索引为: %d\n",m,n);
16     return 0;
17 }
18 int max(int x,int y){
19     return (x>y)?x:y;
20 }


```

运行效果如下所示:

```

10  #include<stdio.h>
11  int main(){
12      int max(int x,int y);
13      int arr[10],m,n,i;
14      printf("请输入10个数字: ");
15      for (int i = 0; i < 10; i++) {
16          scanf("%d",&arr[i]);
17      }
18      for (i = 1,m = arr[0],n = 0; i < 10; i++) {
19          if(max(m,arr[i])>m){
20              m = max(m, arr[i]);
21              n = i;
22          }
23      }
24      printf("10个数字中最大者为: %d,其下标索引为: %d\n",m,n);
25      return 0;
26  }
27  int max(int x,int y){
28      return (x>y)?x:y;
29  }
30

```

☐ 
  
 请输入10个数字: 1 2 3 4 5 6 7 8 9 0
   
 10个数字中最大者为: 9,其下标索引为: 8
   
 Program ended with exit code: 0



## 7.7.2 数组名作函数参数

除了可以用数组元素作为函数参数外，还可以用数组名作为函数参数（包括实参和形参），应当注意的是：用数组元素作实参时，向形参变量传递的是数组元素的值，而用数组名作为函数实参时，向形参（数组名或指针变量）传递的是数组首元素的地址。

**例7.9** 有一个一维数组，存放10个学生的成绩，求成绩的平均数。

```
1  #include<stdio.h>
2  int main(){
3      float average(float array[10]);
4      float score[10],aver;
5      printf("请输入10个学生的成绩: ");
6      for (int i = 0; i < 10; i++) {
7          scanf("%f",&score[i]);
8      }
9      aver = average(score);
10     printf("平均分为%.2f\n",aver);
11     return 0;
12 }
13 float average(float array[10]){
14     float aver,sum = array[0];
15     for (int i = 1; i < 10; i++) {
16         sum += array[i];
17     }
18     aver = sum/10;
19     return aver;
20 }
```

运行效果如下所示：

```

10 #include<stdio.h>
11 int main(){
12     float average(float array[10]);
13     float score[10],aver;
14     printf("请输入10个学生的成绩: ");
15     for (int i = 0; i < 10; i++) {
16         scanf("%f",&score[i]);
17     }
18     aver = average(score);
19     printf("平均分为%.2f\n",aver);
20     return 0;
21 }
22 float average(float array[10]){
23     float aver,sum = array[0];
24     for (int i = 1; i < 10; i++) {
25         sum += array[i];
26     }
27     aver = sum/10;
28     return aver;
29 }
30
31

```

请输入10个学生的成绩: 10 10 10 10 10 10 10 10 10 10  
 平均分为10.00  
 Program ended with exit code: 0

用数组名作函数实参时，不是把数组元素的值传递给形参，而是把实参数组的首元素的地址传递给形参数组，这样两个数组就占同一块内存单元。如果实参数组为 `a`，形参数组为 `b`，若 `a` 的首元素地址为 1000，则 `b` 数组首元素的地址也是 1000，显然，`a[0]` 与 `b[0]` 同占一个单元...假如改变了 `b[0]` 的值，也就意味着 `a[0]` 的值也改变了，也就是说，形参数组中各元素的值如发生变化会使实参数组元素的值同时发生变化。

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
起始地址1000	2	4	6	8	10	12	14	16	18	20
	b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]

图 7.14

例7.10 用选择法对数组中10个整数按由小到大排序。

```

1  #include<stdio.h>
2  int main(){
3      void select_sort(int array[],int n);
4      int arr[10],i;
5      printf("请输入一个数组: ");
6      for (i = 0; i < 10; i++) {
7          scanf("%d",&arr[i]);

```

```

8     }
9     select_sort(arr, 10);
10    printf("排序之后的数组为: ");
11    for (i = 0; i < 10; i++) {
12        printf("%5d", arr[i]);
13    }
14    printf("\n");
15    return 0;
16 }
17
18 void select_sort(int a[], int n) //n为数组a的元素个数
19 {
20     //进行N-1轮选择
21     for(int i=0; i<n-1; i++)
22     {
23         int min_index = i;
24         //找出第i小的数所在的位置
25         for(int j=i+1; j<n; j++)
26         {
27             if(a[j] < a[min_index])
28             {
29                 min_index = j;
30             }
31         }
32         //将第i小的数，放在第i个位置；如果刚好，就不用交换
33         if( i != min_index)
34         {
35             int temp = a[i];
36             a[i] = a[min_index];
37             a[min_index] = temp;
38         }
39     }
40 }

```

运行结果如下所示：

```

~
9
10 #include<stdio.h>
11 int main(){
12     void select_sort(int array[],int n);
13     int arr[10],i;
14     printf("请输入一个数组: ");
15     for (i = 0; i < 10; i++) {
16         scanf("%d",&arr[i]);
17     }
18     select_sort(arr, 10);
19     printf("排序之后的数组为: ");
20     for (i = 0; i < 10; i++) {
21         printf("%5d",arr[i]);
22     }
23     printf("\n");
24     return 0;
25 }
26
27 void select_sort(int a[],int n)//n为数组a的元素个数

```

```

请输入一个数组: 12 2 1 4 5 32 123 47 86 4
排序之后的数组为: 1 2 4 4 5 12 32 47 86 123
Program ended with exit code: 0

```

### 7.7.3 多维数组名作函数参数

可以用多维数组名作为函数的实参和形参，在被调用函数中对形参数组定义时可以指定每一维的大小，也可以省略第一维的大小说明。

```
1 | int array[3][10] 等价于 int array[][10]
```

这是因为二维数组是由若干个一维数组组成的，在内存中，数组是按行存放的，因此，再定义二维数组时，必须指定列数（即一行中包含几个元素），由于形参数组与实参数组类型相同，所以它们是由具有相同长度的一维数组所组成的，不能只指定第1维而省略第2维。

在第2维大小相同的情况下，形参数组的第1维可以与实参数组不同，如实参数组定义为 `int score[5][10]`，而形参数组定义为 `int array[][10]`。

c语言编译系统不检查第1维的大小。

#### 例7.11 求二维数组中的最大值

```

1 | #include<stdio.h>
2 | int main(){
3 |     int max_value(int array[3][3]);
4 |     int arr[3][3]={0},{0},{0};
5 |     printf("请输入原数组元素: ");
6 |     for (int i = 0; i < 3; i++) {
7 |         for (int j = 0; j < 3; j++) {

```

```

8         scanf("%d",&arr[i][j]);
9     }
10 }
11 printf("原数组为: \n");
12 for (int i = 0; i < 3; i++) {
13     for (int j = 0; j < 3; j++) {
14         printf("%3d",arr[i][j]);
15     }
16     printf("\n");
17 }
18 printf("原数组中的最大值为:%d\n",max_value(arr));
19 return 0;
20 }
21 int max_value(int array[3][3]){
22     int i ,j ,max;
23     max = array[0][0];
24     for (i = 0; i < 3; i++) {
25         for (j = 0; j < 3; j++) {
26             if(array[i][j]>max){
27                 max = array[i][j];
28             }
29         }
30     }
31     return max;
32 }

```


运行结果为：

```

9
10 #include<stdio.h>
11 int main(){
12     int max_value(int array[3][3]);
13     int arr[3][3]={0},{0},{0};
14     printf("请输入原数组元素: ");
15     for (int i = 0; i < 3; i++) {
16         for (int j = 0; j < 3; j++) {
17             scanf("%d",&arr[i][j]);
18         }
19     }
20     printf("原数组为: \n");
21     for (int i = 0; i < 3; i++) {
22         for (int j = 0; j < 3; j++) {
23             printf("%3d",arr[i][j]);
24         }
25         printf("\n");
26     }
27     printf("原数组中的最大值为:%d\n",max_value(arr))
28     return 0;
29 }
30 int max_value(int array[3][3]){
31     int i ,j ,max;
32     max = array[0][0];
33     for (i = 0; i < 3; i++) {

```

```

☐ 
请输入原数组元素: 1 2 3 4 5 6 7 8 9
原数组为:
  1  2  3
  4  5  6
  7  8  9
原数组中的最大值为:9
Program ended with exit code: 0

```

## 7.8 局部变量和全局变量

### 7.8.1 局部变量

定义变量可能有三种情况:

- 在函数的开头定义;
- 在函数内的复合语句内定义;
- 在函数的外部定义;

局部变量概念: 在函数或复合语句内部定义的变量。

```

float f1(int a)                                //定义函数 f1
{
    {int b,c; }                               //在函数 f1 中定义 b,c
    :
}
a,b,c 有效

char f2(int x,int y)                           //定义函数 f2
{
    {int i,j; }
    :
}
x,y,i,j 有效

int main()                                     //主函数
{
    {int m,n; }
    :
    return 0;
}
m,n 有效

```

「注」：

- 主函数中定义的变量也只在主函数中有效，并不因为在主函数中定义而在整个文件或程序中有效，主函数也不能使用其他函数中定义的变量；
- 不同函数中可以使用同名的变量，它们代表不同的对象，互不干扰；
- 形式参数也是局部变量；
- 在一个函数内部，可以在复合语句中定义变量，这些变量只在本复合语句中有效，这种语句也称为分程序或程序块；

```

int main ()
{
    { int a,b;
      :
      { int c;
        c=a+b;
        :
      }
      :
    }
}
c 在此复合语句内有效
a,b 在此范围内有效

```

变量 **c** 只在复合语句（分程序）内有效，离开该复合语句该变量就无效，系统会把它占用的内存单元释放。

## 7.8.2 全局变量

概念：在函数内定义的变量是局部变量，在函数外定义的变量是全局变量。

```

int p=1,q=5;                //定义外部变量
float f1(int a)              //定义函数 f1
{
    int b,c;                //定义局部变量
    :
}
char c1,c2;                 //定义外部变量
char f2 (int x, int y)      //定义函数 f2
{
    int i,j;
    :
}
int main()                  //主函数
{
    int m,n;
    :
    return 0;
}

```

全局变量 p,q 的作用范围

全局变量 c1,c2 的作用范围

设置全局变量的作用是增加了函数间数据联系的渠道，由于同一文件中的所有函数都能引用全局变量的值，因此如果在某一个函数中改变了全局变量的值，就能影响到其他函数中全局变量的值。相当于各个函数间有直接的传递通道，由于函数的调用只能带回一个函数返回值，因此有时可以利用全局变量来增加函数间的联系渠道，通过函数调用能得到一个以上的值。

**例 7.12** 编写函数求10个学生成绩的平均分、最高分和最低分。

```

1  #include<stdio.h>
2  float Max = 0.0,Min = 0.0;
3  int main(){
4      float average(float array[],int n);
5      float ave,score[10];
6      printf("请输入学生成绩: ");
7      for (int i = 0; i < 10; i++) {
8          scanf("%f",&score[i]);
9      }
10     ave = average(score, 10);
11     printf("max=%.2f,min=%.2f,average=%.2f\n",Max,Min,ave);
12     return 0;
13 }
14 float average(float array[],int n){
15     float aver,sum = array[0];
16     Max = Min = array[0];
17     for (int i = 1; i < n; i++) {
18         if(array[i]>Max){
19             Max = array[i];
20         }else if (array[i]<Min){
21             Min = array[i];
22         }

```



```

23     sum += array[i];
24 }
25 aver = sum/n;
26 return aver;
27 }

```

运行效果如下所示：

```

10 #include<stdio.h>
11 float Max = 0.0,Min = 0.0;
12 int main(){
13     float average(float array[],int n);
14     float ave,score[10];
15     printf("请输入学生成绩: ");
16     for (int i = 0; i < 10; i++) {
17         scanf("%f",&score[i]);
18     }
19     ave = average(score, 10);
20     printf("max=%.2f,min=%.2f,average=%.2f\n",Max,Min,ave);
21     return 0;
22 }
23 float average(float array[],int n){
24     float aver,sum = array[0];
25     Max = Min = array[0];
26     for (int i = 1; i < n; i++) {
27         if(array[i]>Max){
28             Max = array[i];
29         }else if (array[i]<Min){
30             Min = array[i];
31         }
32         sum += array[i];
33     }
34     aver = sum/n;

```

```

请输入学生成绩: 1 2 3 4 5 6 7 8 9 10
max=10.00,min=1.00,average=5.50
Program ended with exit code: 0

```

一般情况下，不建议使用全局变量：

- 全局变量在程序的执行过程中都占用内存单元，而不是仅在需要时才开辟单元；
- 降低了函数的通用型，当将该函数移植到另一个文件中时，还要考虑把相关的外部变量一起移植过去，但是若该外部变量与其他文件的变量同名时，就会出现问題，程序设计要求单个模块内部内聚性强，与其他模块耦合性弱；
- 使用全局变量过多，会降低程序的清晰性，人们往往难以清除地判断出每个瞬时各个外部变量的值；

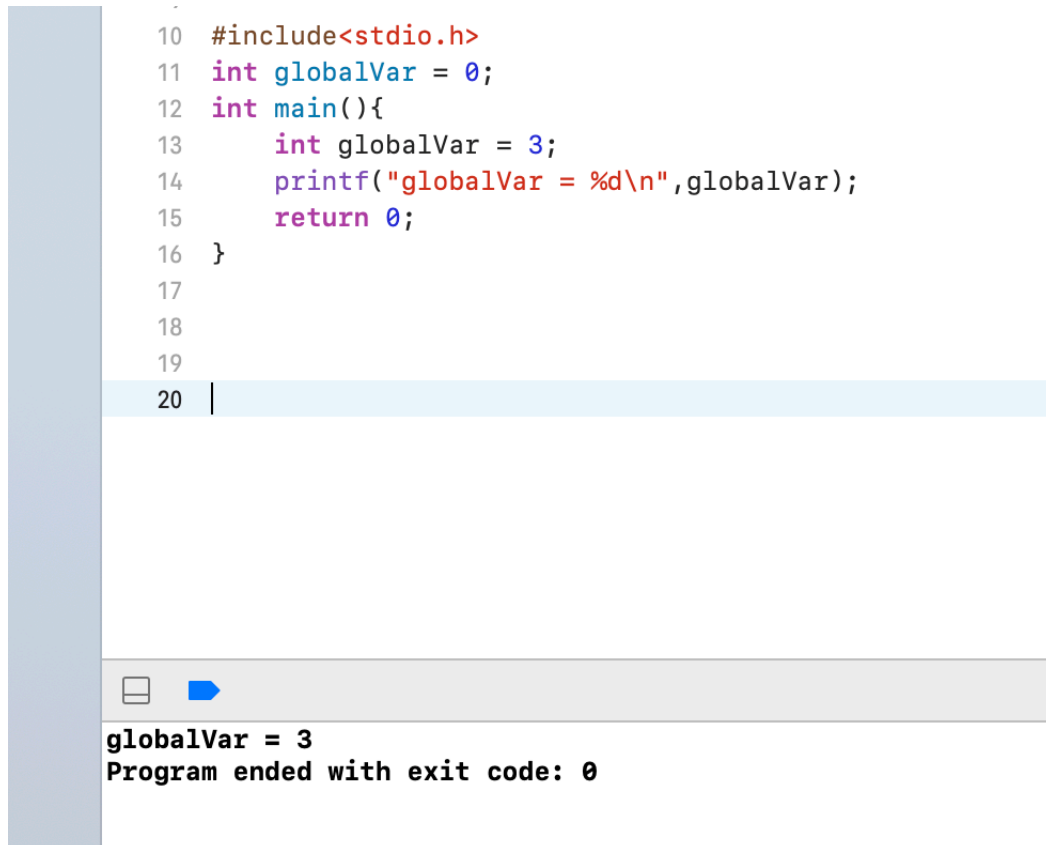
例 7.13 全局变量与局部变量同名。

```

1  #include<stdio.h>
2  int globalVar = 0;
3  int main(){
4      int globalVar = 3;
5      printf("globalVar = %d\n",globalVar);
6      return 0;
7  }

```

运行效果如下所示：



```

10  #include<stdio.h>
11  int globalVar = 0;
12  int main(){
13      int globalVar = 3;
14      printf("globalVar = %d\n",globalVar);
15      return 0;
16  }
17
18
19
20

```

globalVar = 3  
Program ended with exit code: 0

## 7.9 变量的存储方式和生存期

### 7.9.1 动态存储方式与静态存储方式

变量的存储有两种不同的方式：**静态存储方式**和**动态存储方式**，静态存储方式是指在程序运行期间由系统分配固定的存储空间的方式，而动态存储方式则是在程序运行期间根据需要进行动态的分配存储空间的方式。

可供用户使用的存储空间分为三部分：程序区、静态存储区、动态存储区。

数据分别存放在静态存储区和动态存储区，全局变量全部存放在静态存储区中，在程序开始执行时给全局变量分配存储区，程序执行完毕就释放，在程序执行过程中它们占据固定的存储单元，而不是动态地进行分配和释放。

在动态存储区中存放以下数据：

- 函数形式参数，在函数调用时给形参分配存储空间；

- 函数中定义的没有用关键字 `static` 声明的变量，即自动变量；
- 函数调用时的现场保护和返回地址；

对以上这些数据，在函数调用开始时分配动态存储空间，函数结束时释放这些空间。在程序执行过程中，这种分配和释放是动态的，如果在一个程序中两次调用同一函数，而在此函数中定义了局部变量，在两次调用时分配给这些局部变量的存储空间的地址可能是不相同的。

C 语言中，每一个变量和函数都有两个属性：数据类型和数据的存储类别，存储类别指的是数据在内存中存储的方式，如静态存储和动态存储，再定义和声明变量和函数时，一般应同时指定其数据类型和存储类别，也可以采用默认方式指定，其存储类别包括 `auto`、`static`、`register`、`extern` 四种。

## 7.9.2 局部变量的存储类别

### 1. 自动变量 `auto`

函数中的局部变量，如果不专门声明为 `static` 存储类别，都是动态地分配存储空间的，数据存储在动态存储区中。函数中的形参和在函数中定义的局部变量（包括在复合语句中定义的局部变量）都属于此类，调用该函数时，系统会自动给这些变量分配存储空间，在函数调用时就自动释放这些存储空间，因此这类局部变量称为自动变量，通过关键字 `auto` 进行声明，实际上，该关键字可以省略不写，不写 `auto` 则隐含指定为自动存储类别，它属于动态存储方式。

```
1  int b,c = 3;
2  与
3  auto int b,c = 3;等价
```

### 2. 静态局部变量 `static`

有时希望函数中的局部变量的值在函数调用结束后不消失而继续保留原有值，即其占用的存储单元不释放，在下次调用该函数时，该变量已有值，即上一次函数调用结束时的值，此时应通过 `static` 关键字进行声明。

**例 7.14** 考察静态局部变量的值。

```
1  #include<stdio.h>
2  int main(){
3      int f(int n);
4      int a = 2;
5      for (int i = 0; i < 3;i++) {
6          printf("%d\n",f(a));
7      }
8      return 0;
9  }
10 int f(int a){
11     auto int b = 0;
12     static int c = 3;
13     b += 1;
14     c += 1;
15     return a+b+c;
16 }
```

运行结果如下所示：

```
10 #include<stdio.h>
11 int main(){
12     int f(int n);
13     int a = 2;
14     for (int i = 0; i < 3;i++) {
15         printf("%d\n",f(a));
16     }
17     return 0;
18 }
19 int f(int a){
20     auto int b = 0;
21     static int c = 3;
22     b += 1;
23     c += 1;
24     return a+b+c;
25 }
26
27
28
```

7  
8  
9  
Program ended with exit code: 0

「注」：

- 对静态局部变量是在编译时赋初值的，即只赋初值一次，在程序运行时它已有初值，以后每次调用函数时不在重新赋初值而只是保留上次函数调用结束时的值，而对自动变量赋初值，不是在编译时进行的，而是在函数调用时进行的，每调用一次函数重新给一次初值，相当于执行一次赋值语句；
- 如果在定义局部变量时不赋初值的话，则对静态局部变量来说，编译时自动赋初值0（对数值型变量）或空字符 \0（对字符变量），而对自动变量来说，它的值是一个不确定值，这是由于每次函数调用结束后存储单元已经释放，下次调用时又重新分配存储单元，而所分配的单元中的内容是不可知的。



例 7.15 求阶乘。

```
1  #include<stdio.h>
2  int main(){
3      int fac(int n);
4      for (int i = 1; i <= 5; i++) {
5          printf("%d!=%d\n",i,fac(i));
6      }
7      return 0;
8  }
9
10 int fac(int n){
11     static int f = 1;
```

```
12     f = f * n;
13     return f;
14 }
```

运行效果如下所示：

```
9
10 #include<stdio.h>
11 int main(){
12     int fac(int n);
13     for (int i = 1; i <= 5; i++) {
14         printf("%d!=%d\n",i,fac(i));
15     }
16     return 0;
17 }
18
19 int fac(int n){
20     static int f = 1;
21     f = f * n;
22     return f;
23 }
24
25 |
26
```

```
1!=1
2!=2
3!=6
4!=24
5!=120
Program ended with exit code: 0
```

「注」：静态存储要长期占用内存，故不建议多用静态变量。

### 3. 寄存器变量

一般情况下，变量的值是存放在内存中的，当程序中用到哪一个变量的值时，由控制器发出指令将内存中该变量的值送到运算器中，经过运算器进行运算，如果需要存数，再从运算器将数据送到内存存放。

如果有一些变量使用频繁，则为存取变量的值需要花费不少时间，为提高效率，运行将局部变量的值放在CPU中的寄存器中，需要用时直接从寄存器取出参加运算，不必在到内存中去存取，由于对寄存器的存取速度远高于对内存的存取速度，因此这样做可以提高执行效率，这种变量叫做寄存器变量，用关键字 `register` 做声明。

然而，由于现在计算的速度越来越快，性能越来越高，因此目前基本不在使用 `register` 变量。

### 7.9.3 全局变量的存储类别

全局变量都是存放在静态存储区的，因此它的生存期是固定的，存在于程序的整个运行过程。一般来说，外部变量是在函数的外部定义的全局变量，它的作用域是从变量的定义处开始，到本程序文件的末尾。在此作用域内，全局变量可以为程序中各个函数所引用。

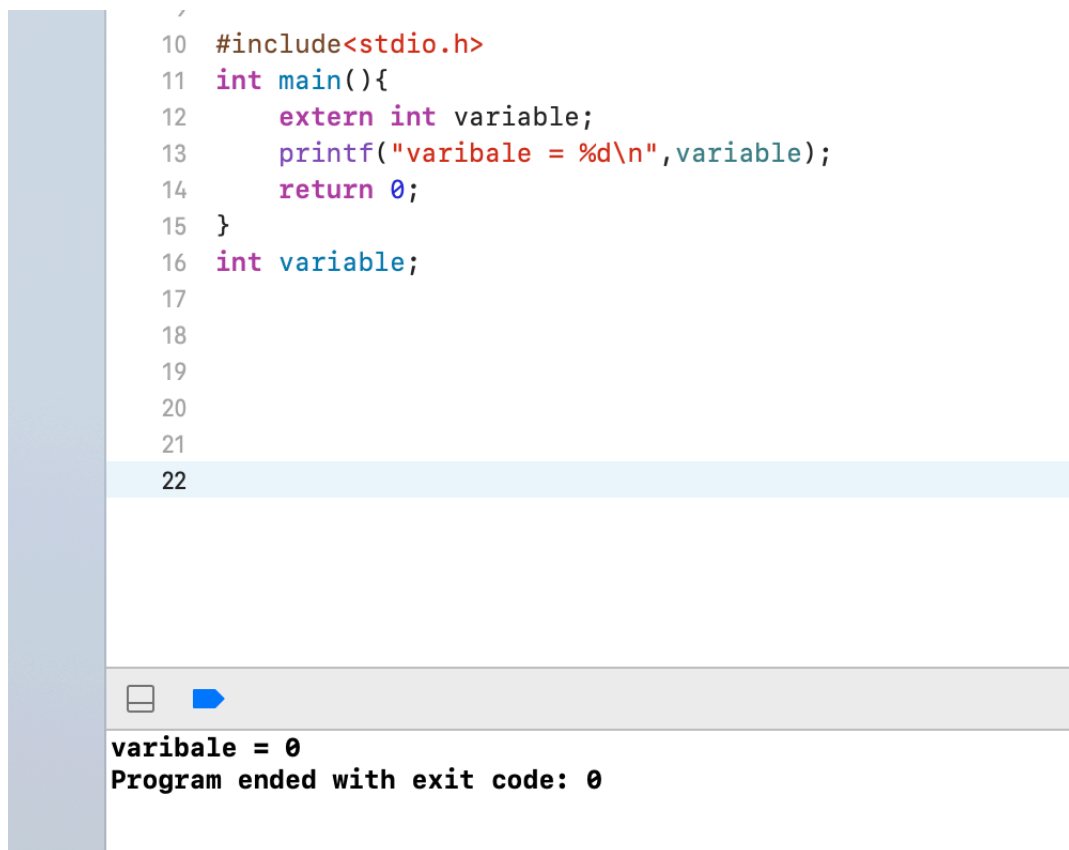
以下是扩展外部变量作用域的三种情况：

- 在一个文件内扩展外部变量的作用域

通过 `extern` 对该变量做外部声明，表示将该外部变量的作用域扩展到此为止。

```
1  #include<stdio.h>
2  int main(){
3      extern int variable;
4      printf("varibale = %d\n",variable);
5      return 0;
6  }
7  int variable;
```

运行效果如下所示：



The screenshot shows a code editor with a light blue sidebar on the left. The code is as follows:

```
10 #include<stdio.h>
11 int main(){
12     extern int variable;
13     printf("varibale = %d\n",variable);
14     return 0;
15 }
16 int variable;
17
18
19
20
21
22
```

Below the code editor, there is a terminal window with a blue arrow icon on the left. The output of the program is:

```
varibale = 0
Program ended with exit code: 0
```

「注」：提倡将外部变量的定义放在引用它的所有函数之前，这样可以避免在函数中多加一个 `extern` 声明。

- 将外部变量的作用域扩展到其他文件

如果一个程序中包含两个文件，在两个文件中都要用到同一个外部变量时，正确的做法是：在任一个文件中定义外部变量，然后在另一文件中有 `extern` 对该外部变量做声明即可。

```

文件 file1.c:

#include <stdio.h>
int A;                                //定义外部变量
int main()
{
    int power(int);                    //函数声明
    int b=3,c,d,m;
    printf("enter the number a and its power m:\n");
    scanf("%d,%d",&A,&m);
    c=A*b;
    printf("%d * %d = %d\n",A,b,c);
    d=power(m);
    printf("%d * %d = %d\n",A,m,d);
    return 0;
}

文件 file2.c:

extern A;                             //把在 file1 文件中已定义的外部变量的作用域扩展到本文件
int power(int n)
{
    int i,y=1;
    for(i=1;i<=n;i++)
        y*=A;
    return(y);
}

```

实际上，在编译时遇到 `extern` 时，先在本文件中找外部变量的定义，如果找到，就在本文件中扩展作用域，如果找不到，就在连接时从其他文件中找外部变量的定义，如果从其他文件中找到了，就将作用域扩展到本文件，如果再找不到，就按出错处理。

- 将外部变量的作用域限制在本文件中

有时在程序设计中希望某些外部变量只限于被本文件引用，而不能被其他文件引用，这时可以在定义外部变量时加一个 `static` 声明。

用 `static` 声明一个变量的作用是：

1. 对局部变量用 `static` 声明，把它分配在静态存储区，该变量在整个程序执行期间不释放，其所分配的空间始终存在；
2. 对全局变量用 `static` 声明，则该变量的作用域只限于本文件模块；

## 7.10 关于变量的声明和定义

对变量而言，声明与定义的关系稍微复杂一点，在声明部分出现的变量有两种情况，一种是需要建立存储空间（如 `int a;`），另一种是不需要建立存储空间的（如 `extern a;`），前者称为定义性声明，或简称**定义**，后者称为引用性声明。为了叙述方便，把建立存储空间的声明称为定义，而把不需要建立存储空间的声明称为声明。

```

int main()
{
    extern A          //是声明,不是定义。声明将已定义的外部变量 A 的作用域扩展到此
    :
    return 0;
}

int A;               //是定义,定义 A 为整型外部变量

```

## 7.11 内部函数和外部函数

函数本质上是全局的，因为定义一个函数的目的就是要被另外的函数调用，如果不加声明的话，一个文件中的函数既可以被本文件中其他函数调用，也可以被其他文件中的函数调用。但是，也可以指定某些函数不能被其他文件调用，根据函数能否被其他源文件调用，将函数区分为内部函数和外部函数。

### 7.11.1 内部函数

如果一个函数只能被本文件中其他函数所调用，它称为内部函数，在定义内部函数时，在函数名和函数类型前加上 `static` 关键字即可，其定义形式为：

```
1 | static int fun(int a,int b);
```

### 7.11.2 外部函数

如果在定义函数时，在函数首部的最左端加关键字 `extern`，则此函数是外部函数，可供其他文件调用，C 语言规定，如果在定义函数时省略 `extern`，则默认为外部函数。

在需要调用此函数的其他文件中，需要对此函数作声明，在对此函数做声明时，要加关键字 `extern`，表示该函数是在其他文件中定义的外部函数，其定义形式为：

```
1 | extern int fun(int a,int b)
```

课后题：1、3、4、5、6、9、10、17