

第 7 章

用函数实现模块化程序设计

为什么要用函数

```
int main()
{
    {
        : 功能1内容
    }

    {
        : 功能2内容
    }

    {
        : 功能1内容
    }

    {
        : 功能2内容
    }
}
```

VS.

```
功能1函数()
{
    : 功能1内容
}

功能2函数()
{
    : 功能2内容
}

int main()
{
    调用功能1
    调用功能2
    调用功能1
    调用功能2
}
```

- 使用函数可使程序清晰、精炼、简单、灵活。
- 函数就是功能。每一个函数用来实现一个特定的功能。函数名应反映其代表的功能。
- 在设计较大程序时，往往把它分为若干个程序模块，每一个模块包括一个或多个函数，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。

为什么要用函数

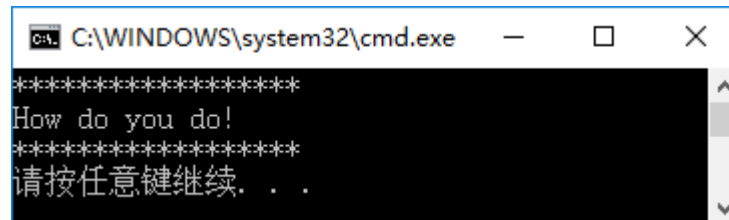
【例7.1】想输出以下的结果，用函数调用实现。

```
#include <stdio.h>

int main()
{
    void print_star();           //声明print_star函数
    void print_message();        //声明print_message函数
    print_star();                //调用print_star函数
    print_message();             //调用print_message函数
    print_star();                //调用print_star函数
    return 0;
}

void print_star()                //定义print_star函数
{
    printf("*****\n");          //输出一行*号
}

void print_message()             //定义print_message函数
{
    printf("How do you do!\n");  //输出一行文字信息
}
```



```
C:\WINDOWS\system32\cmd.exe
*****
How do you do!
*****
请按任意键继续...
```



print_star和print_message都是用户定义的函数名，分别用来输出一排“*”号和一行文字信息。在定义这两个函数时指定函数的类型为void，意为函数无类型，即无函数值，也就是说，执行这两个函数后不会把任何值带回main函数。

在程序中，定义print_star函数和print_message函数的位置是在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)。

为什么要用函数

(1) 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。较大的程序，可分别放在若干个源文件中。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。

(2) 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。

(3) 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，在调用后流程返回到main函数，在main函数中结束整个程序的运行。

(4) 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。

(5) 从用户使用的角度看，函数有两种。

① **库函数**，它是由系统提供的，用户不必自己定义，可直接使用它们。应该说明，不同的C语言编译系统提供的库函数的数量和功能会有一些不同，当然许多基本的函数是共同的。

② **用户自己定义的函数**。它是用以解决用户专门需要的函数。

(6) 从函数的形式看，函数分两类。

① **无参函数**。在调用无参函数时，主调函数不向被调用函数传递数据。

② **有参函数**。在调用函数时，主调函数在调用被调用函数时，通过参数向被调用函数传递数据。

定义函数

为什么定义函数

C语言要求，在程序中用到的所有函数，必须“先定义，后使用”。

定义函数应包括以下几个内容：

- (1) 指定函数的**名字**，以便以后按名调用。
 - (2) 指定函数的**类型**，即函数返回值的类型。
 - (3) 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。
 - (4) 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**。这是最重要的，是在函数体中解决的。
-

说明: `int max(int x, int y);`

调用:

```
int a, b, c;
```

```
c = max(a, b);
```

定义: `int max(int x, int y) { }`

定义函数的方法

定义无参函数

```
类型名  函数名()  
{  
    函数体  
}
```

或

```
类型名  函数名(void)  
{  
    函数体  
}
```

函数名后面括号内的void表示“空”，即函数没有参数。

函数体包括声明部分和语句部分。

在定义函数时要用“类型标识符”（即类型名）指定函数值的类型，即指定函数带回来的值的类型。

定义有参函数

```
类型名  函数名(形式参数表列)  
{  
    函数体  
}
```

```
int max(int x, int y)  
{  
    int z;           //声明部分  
    z=x>y?x:y;       //执行语句部分  
    return(z);  
}
```

定义空函数

```
类型名  函数名()  
{ }
```

函数体为空，什么也不做。

调用函数

函数调用的形式

函数名(实参表列)

```
int max(int x, int y) { }
```

```
printf_star(); //调用无参函数  
c = max(a, b); //调用有参函数
```

1. 函数调用语句

把函数调用单独作为一个语句。如`printf_star()`;

这时不要求函数带返回值，只要求函数完成一定的操作。

2. 函数表达式

函数调用出现在另一个表达式中，如`c = max(a, b)`;

这时要求函数带回一个确定的值以参加表达式的运算。

3. 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))`;又如:`printf ("%d", max(a, b))`;



形式参数和实际参数

说明: `int max(int x, int y);`

调用:

```
int a, b, c;
```

```
c = max(a, b);
```

定义: `int max(int x, int y) { }`

在调用有参函数时，主调函数和被调用函数之间有数据传递关系。

在定义函数时函数名后面括号中的变量名称为“**形式参数**”（简称“**形参**”）或“**虚拟参数**”。

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**”（简称“**实参**”）。实际参数可以是常量、变量或表达式，但要求它们有确定的值。

实参与形参的类型应相同或赋值兼容。赋值兼容是指实参与形参类型不同时能按不同类型数值的赋值规则进行转换。

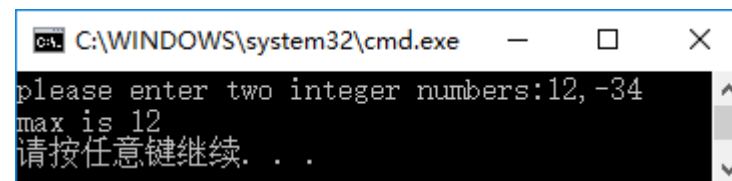


实参和形参间的数据传递

【例7.2】输入两个整数，要求输出其中值较大者。要求用函数来找到大数。

```
#include <stdio.h>
int main()
{   int max(int x,int y);           //对max函数的声明
    int a,b,c;
    printf("please enter two integer numbers:"); //提示
    scanf("%d,%d",&a,&b);          //输入两个整数
    c = max(a,b); //调用max函数，有两个实参。大数赋给变量c
    printf("max is %d\n",c); //输出大数c
    return 0;
}
```

```
int max(int x,int y) //定义max函数，有两个参数
{
    int z;           //定义临时变量z
    z=x>y?x:y;       //把x和y中大者赋给z
    return(z);        //把z作为max函数的值带回main函数
}
```



```
C:\WINDOWS\system32\cmd.exe
please enter two integer numbers:12,-34
max is 12
请按任意键继续. . .
```



定义函数，名为max，函数类型为int。指定两个形参x和y，形参的类型为int。

主函数中包含了一个函数调用max(a,b)。max后面括号内的a和b是实参。a和b是在main函数中定义的变量，x和y是函数max的形式参数。通过函数调用，在两个函数之间发生数据传递，实参a和b的值传递给形参x和y，在max函数中把x和y中的大者赋给变量z，z的值作为函数值返回main函数，赋给变量c。

c=max(a,b); (main函数)

int max(int x,int y) (max函数)
{ int z;
 z=x>y?x:y;
 return(z);
}

在调用函数过程中发生的实参与形参间的数据传递称为“虚实结合”。

函数调用的过程

说明: `int max(int x, int y);`

调用:

```
int a, b, c;
```

```
c = max(a, b);
```

定义: `int max(int x, int y) { }`

- (1) 在定义函数中指定的**形参**，在未出现函数调用时，它们并不占内存中的存储单元。在发生函数调用时，函数的形参才被临时分配内存单元。
- (2) 将实参的值传递给对应形参。
- (3) 在执行函数期间，由于形参已经有值，就可以利用形参进行有关的运算。
- (4) 通过return语句将函数值带回到主调函数。应当注意返回值的类型与函数类型一致。如果函数不需要返回值，则不需要return语句。这时函数的类型应定义为void类型。
- (5) 调用结束，形参单元被释放。注意：**实参单元仍保留并维持原值，没有改变**。如果在执行一个被调用函数时，形参的值发生改变，不会改变主调函数的实参的值。因为**实参与形参是两个不同的存储单元**。

注意

- 实参向形参的数据传递是“**值传递**”，**单向传递**，只能由实参传给形参，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。

函数的返回值

```
int max(float x, float y);           //函数值为整型
char letter(char c1, char c2);       //函数值为字符型
double min(int x, int y);            //函数值为双精度型
void fun(int x, int y); // 无返回值
```

通常，希望通过函数调用使主调函数能得到一个确定的值，这就是**函数值**(函数的返回值)。

(1) 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，执行到哪一个return语句，哪一个return语句就起作用。return语句后面的括号可以不要，如“return z;”与“return(z);”等价。return后面的值可以是一个表达式。

(2) 函数值的类型。函数值的类型在定义函数时指定。

(3) 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。

如果函数值的类型和return语句中表达式的值不一致，则以函数类型为准。对数值型数据，可以自动进行类型转换。即**函数类型决定返回值的类型**。

(4) 对于不带返回值的函数，应当用定义函数为“**void类型**”（或称“空类型”）。这样，系统就保证不使函数带回任何值，即禁止在调用函数中使用被调用函数的返回值。此时在函数体中不得出现return语句。

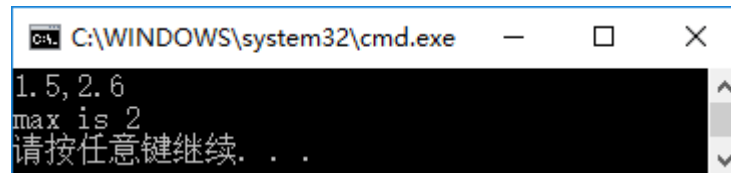
函数的返回值

【例7.3】将例7.2稍作改动，将在max函数中定义的变量z改为float型。


函数返回值的类型与指定的函数类型不同，分析其处理方法。

```
#include <stdio.h>
int main()
{
    int max(float x, float y);
    float a, b;
    int c;
    scanf("%f, %f", &a, &b);
    c = max(a, b);
    printf("max is %d\n", c);
    return 0; }
```

```
int max(float x, float y)
{
    float z;    //z为实型变量
    z = x > y ? x : y;
    return(z);
}
```



```
C:\WINDOWS\system32\cmd.exe
1.5, 2.6
max is 2
请按任意键继续. . .
```

 max函数的形参是float型，实参也是float型，在main函数中输入给a和b的值是1.5和2.6。在调用max(a, b)时，把a和b的值1.5和2.6传递给形参x和y。执行函数max中的条件表达式“z=x>y?x:y”，使得变量z得到的值为2.6。现在出现了矛盾：函数定义为int型，而return语句中的z为float型，要把z的值作为函数的返回值，二者不一致。怎样处理呢？按赋值规则处理，先将z的值转换为int型，得到2，它就是函数得到的返回值。如果将main函数中的c改为float型，用%f格式符输出，输出2.000000。因为调用max函数得到的是int型，函数值为整数2。

对被调用函数的声明和函数原型

在一个函数中调用另一个函数（即被调用函数）需要具备如下条件：

- (1) 首先被调用的函数必须是已经定义的函数（是库函数或用户自己定义的函数）。
- (2) 如果使用库函数，应该在本文件开头用#include指令将调用有关库函数时所需用到的信息“包含”到本文件中来。
- (3) 如果使用用户自己定义的函数，而该函数的位置在调用它的函数（即主调函数）的后面（在同一个文件中），应该在主调函数中对被调用的函数作声明(declaration)。声明的作用是把函数名、函数参数的个数和参数类型等信息通知编译系统，以便在遇到函数调用时，编译系统能正确识别函数并检查调用是否合法。

说明: `int max(int x, int y);`

调用:

```
int a, b, c;  
c = max(a, b);
```

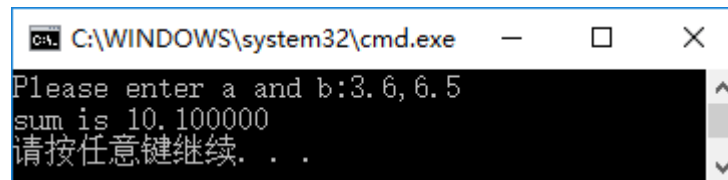
定义: `int max(int x, int y) { }`

对被调用函数的声明和函数原型

【例7.4】输入两个实数，用一个函数求出它们之和。

```
#include <stdio.h>
int main()
{   float add(float x, float y);    //对add函数作声明
    float a, b, c;
    printf("Please enter a and b:"); //提示输入
    scanf("%f, %f", &a, &b);        //输入两个实数
    c=add(a, b);                    //调用add函数
    printf("sum is %f\n", c);        //输出两数之和
    return 0;
}
```

```
float add(float x, float y)    //定义add函数
{   float z;
    z=x+y;
    return(z);                //把变量z的值作为函数值返回
}
```



```
C:\WINDOWS\system32\cmd.exe
Please enter a and b:3.6, 6.5
sum is 10.100000
请按任意键继续. . .
```

 函数的声明和函数定义中的第1行（函数首部）基本上是相同的，只差一个分号（函数声明比函数定义中的首行多一个分号）。函数的首行（即函数首部）称为**函数原型**（function prototype）。因为在函数的首部包含了检查调用函数是否合法的基本信息（它包括了函数名、函数值类型、参数个数、参数类型和参数顺序），因此，在函数调用时检查函数原型是否与函数声明一致。这样就能保证函数的正确调用。

对被调用函数的声明和函数原型

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);

或 函数类型 函数名(参数类型1, 参数类型2, ..., 参数类型n);

在函数声明中的形参名可以省写，而只写形参的类型。

```
float add(float x, float y);  
float add(float, float);           //不写参数名，只写参数类型  
float add(float a, float b);       //参数名不用x,y，而用a,b。合法
```

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); float f(float, float); int i(float, float);  
//所有函数之前，且在函数外部进行函数声明  
int main() { ... }  
//在main函数中要调用letter, f和i函数，不必再对所调用的这3个函数进行声明  
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
```

注意

- 对函数的“定义”和“声明”不是同一回事。函数的定义是指对函数功能的确立，包括指定函数名、函数值类型、形参及其类型以及函数体等，它是一个完整的、独立的函数单位。而函数的声明的作用则是把函数的名字、函数类型以及形参的类型、个数和顺序通知编译系统，以便在调用该函数时系统按此进行对照检查，它不包含函数体。

函数的嵌套调用

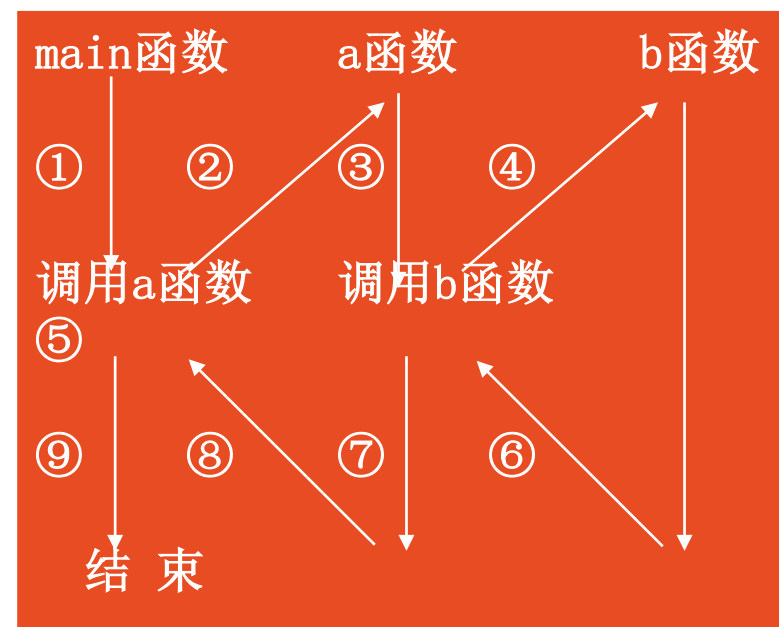
```
int main()
{ ...;
  c = a();
  ...
}
```

```
int a()
{ ...;
  c = b();
  ...
}
```

```
int b()
{ ...;
  return 10;
}
```

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，一个函数内不能再定义另一个函数，即不能嵌套定义，但可以嵌套调用函数，即在调用一个函数的过程中，又调用另一个函数。

- ① 执行main函数的开头部分;
- ② 遇函数调用语句，调用函数a，流程转去a函数;
- ③ 执行a函数的开头部分;
- ④ 遇函数调用语句，调用函数b，流程转去函数b;
- ⑤ 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;
- ⑥ 返回到a函数中调用b函数的位置;
- ⑦ 继续执行a函数中尚未执行的部分，直到a函数结束;
- ⑧ 返回main函数中调用a函数的位置;
- ⑨ 继续执行main函数的剩余部分直到结束。



函数的嵌套调用

```
C:\WINDOWS\system32\cmd.exe
Please enter 4 interger numbers:12 45 -6 89
max=89
请按任意键继续. . .
```

【例7.5】输入4个整数，找出其中最大的数。用函数的嵌套调用来处理。

```
#include <stdio.h>

int main()
{
    int max4(int a,int b,int c,int d); //对max4的函数声明
    int a,b,c,d,max;
    printf("Please enter 4 interger numbers:"); //提示输入4个数
    scanf("%d %d %d %d",&a,&b,&c,&d); //输入4个数
    max=max4(a,b,c,d); //调用max4函数，得到4个数中的最大者
    printf("max=%d \n",max); //输出4个数中的最大者
    return 0;
}
```

```
int max4(int a,int b,int c,int d) //定义max4函数
{
    int max2(int a,int b); //对max2的函数声明
```

```
    int m;
    m=max2(a,b); //调用max2函数，得到a和b中的大者，放在m中
    m=max2(m,c); //调用max2函数，得到a,b,c中的大者，放在m中
    m=max2(m,d); //调用max2函数，得到a,b,c,d中的大者，放在m中
    return(m); //把m作为函数值带回main函数
}
```

```
int max2(int a,int b) //定义max2函数
{
    if(a>=b)
        return a; //若a≥b，将a作为函数返回值
    else
        return b; //若a<b，将b作为函数返回值
}
```



在主函数中要调用max4函数，因此主函数的开头要对max4函数作声明。在max4函数中3次调用max2函数，因此在max4函数的开头要对max2函数作声明。由于在主函数中没有直接调用max2函数，因此主函数中不必对max2函数作声明，只须在max4函数中作声明即可。max4函数执行过程：第1次调用max2函数得到的函数值是a和b中的大者，把它赋给变量m，第2次调用max2得到m和c中的大者，也就是a,b,c中的最大者，再把它赋给变量m。第3次调用max2得到m和d中的大者，也就是a,b,c,d中的最大者，再把它赋给变量m。这是一种递推方法，先求出2个数的大者；再以此为基础求出3个数的大者；再以此为基础求出4个数的大者。m的值一次一次地变化，直到实现最终要求。



程序改进

(1) 可以将max2函数的函数体改为只用一个return语句，返回一个条件表达式的值：

```
int max2(int a, int b)           //定义max2函数
{ return(a>=b?a:b); }           //返回条件表达式的值，即a和b中的大者
```

(2) 在max4函数中，3个调用 max2的语句可以用以下一行代替：

```
m=max2(max2(max2(a, b), c), d); //把函数调用作为函数参数
```

甚至可以取消变量m，max4函数可写成

```
int max4(int a, int b, int c, int d)
{   int max2(int a, int b);           //对max2的函数声明
    return max2(max2(max2(a, b), c), d);
}
```

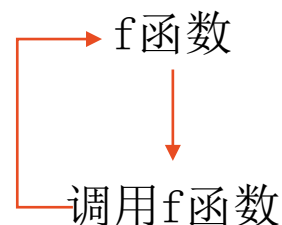
先调用“max2(a, b)”，得到a和b中的大者。再调用“max2(max2(a, b), c)”（其中max2(a, b)为已知），得到a, b, c三者中的大者。最后由“max2(max2(max2(a, b), c), d)”求得a, b, c, d四者中的大者。

函数的递归调用

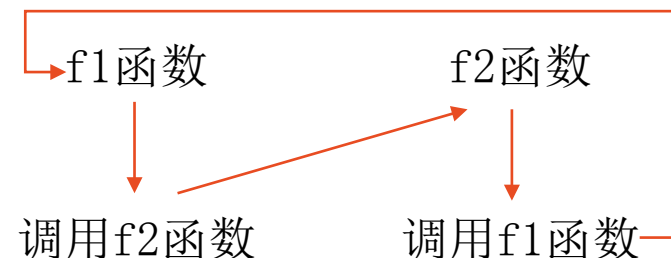
函数的递归调用

在调用一个函数的过程中又出现直接或间接地调用该函数本身，称为函数的递归调用。

```
int f(int x)
{
    int y, z;
    z=f(y); //在执行f函数的过程中又要调用f函数
    return (2*z);
}
```



直接递归



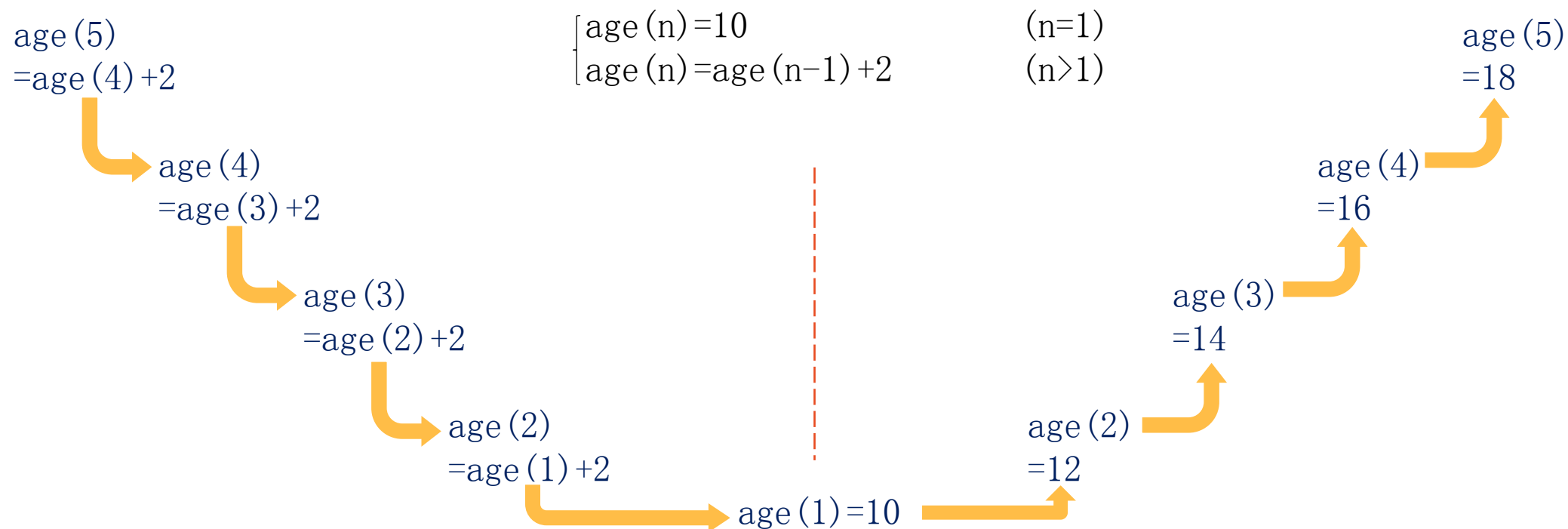
间接递归

程序中不应出现无终止的递归调用，而只应出现有限次数的、有终止的递归调用，这可以用if语句来控制，只有在某一条件成立时才继续执行递归调用；否则就不再继续。

函数的递归调用

【例7.6】有5个学生坐在一起，问第5个学生多少岁，他说比第4个学生大2岁。问第4个学生岁数，他说比第3个学生大2岁。问第3个学生，又说比第2个学生大2岁。问第2个学生，说比第1个学生大2岁。最后问第1个学生， he 说是10岁。请问第5个学生多大。

解题思路：



函数的递归调用

【例7.6】有5个学生坐在一起，问第5个学生多少岁，他说比第4个学生大2岁。问第4个学生岁数，他说比第3个学生大2岁。问第3个学生，又说比第2个学生大2岁。问第2个学生，说比第1个学生大2岁。最后问第1个学生， he说是10岁。请问第5个学生多大。

```
#include <stdio.h>
```

```
int main()
```

```
{    int age(int n);           //对age函数的声明
```

```
    printf("No. 5, age:%d\n", age(5)); //输出第5个学生的年龄
```

```
    return 0;
```

```
}
```

```
int age(int n)
```

```
{    int c;           //定义递归函数
```

```
    if(n==1)         //c用作存放函数的返回值的变量
```

```
        c=10;        //如果n等于1
```

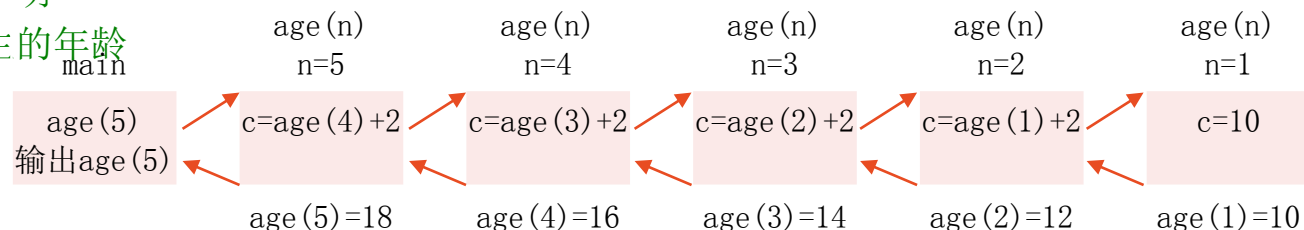
```
        //年龄为10
```

```
    else             //如果n不等于1
```

```
        c=age(n-1)+2; //年龄是前一个学生的年龄加2(如第4个学生年龄是第3个学生年龄加2)
```

```
    return(c);       //返回年龄
```

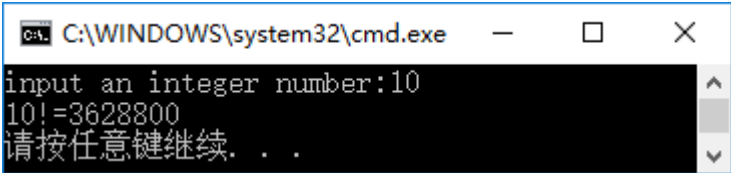
```
}
```



注意分析递归的终止条件。

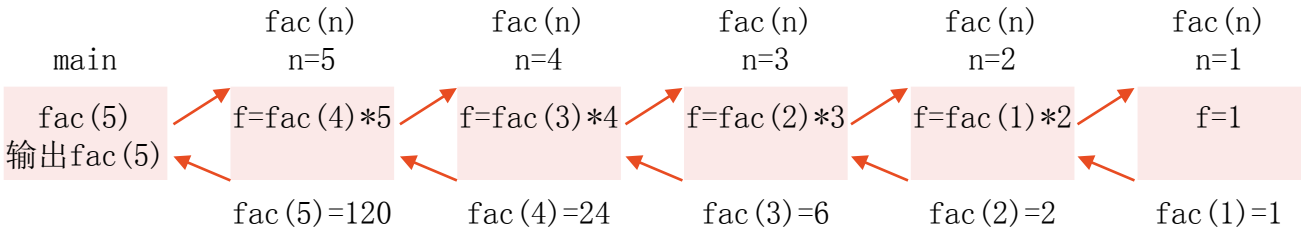
函数的递归调用

【例7.7】用递归方法求n!。



解题思路：

$$n!=\begin{cases} n! = 1 & (n = 0, 1) \\ n \cdot (n - 1)! & (n > 1) \end{cases}$$



```
#include <stdio.h>
int main()
{
    int fac(int n); //fac函数声明
    int n;
    int y;
    printf("input an integer number:");
    scanf("%d",&n); //输入要求阶乘的数
    y=fac(n);
    printf("%d!=%d\n",n,y);
    return 0;
}
```

```
int fac(int n) //定义fac函数
{
    int f;
    if(n<0) //n不能小于0
        printf("n<0,data error!");
    else if(n==0||n==1) //n=0或,1时n!=1
        f=1; //递归终止条件
    else
        f=fac(n-1)*n; //n>1时,n!=n*(n-1)
    return(f);
}
```

注意

• 程序中的变量是int型，如果用Visual C++、GCC以及多数C编译系统为int型数据分配4个字节，能表示的最大数为2 147 483 647，当n=12时，运行正常，输出为479 001 600。如果输入13，企图求13!，是得不到预期结果的，因为求出的结果超过了int型数据的最大值。可将f,y和fac函数定义为float或double型。

函数的递归调用

【例7.8】Hanoi（汉诺）塔问题。古代有一个梵塔，塔内有3个座A,B,C。开始时A座上有64个盘子，盘子大小不等，大的在下，小的在上。有一个老和尚想把这64个盘子从A座移到C座，但规定每次只允许移动一个盘，且在移动过程中在3个座上都始终保持大盘在下，小盘在上。在移动过程中可以利用B座。要求编程输出移动盘子的步骤。

解题思路：

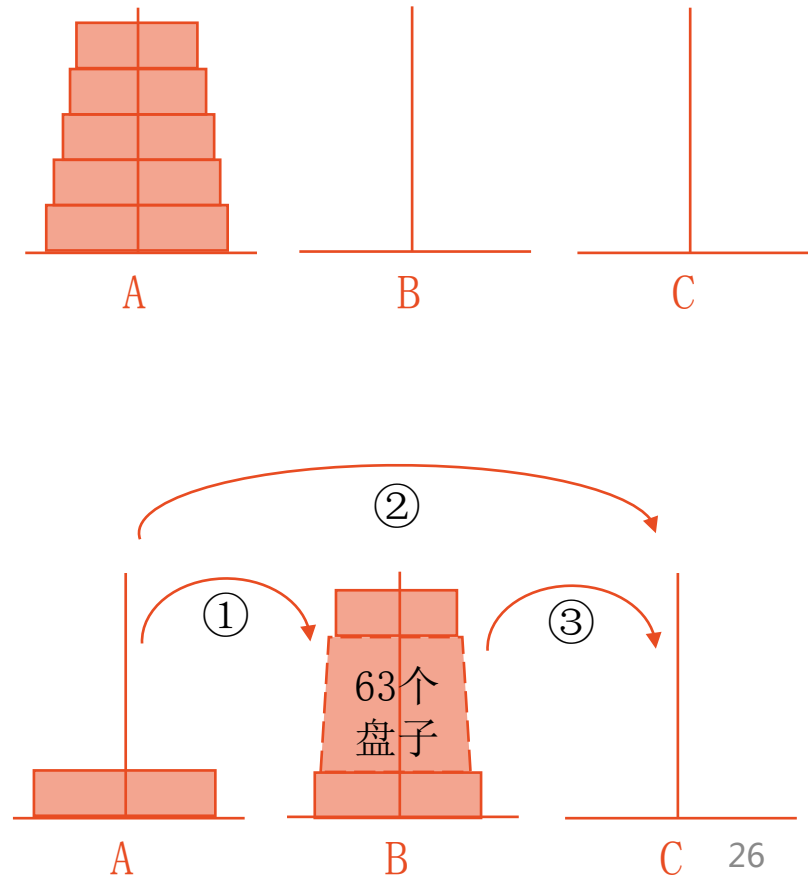
老和尚会这样想：假如有另外一个和尚能有办法将上面63个盘子从一个座移到另一座。那么，问题就解决了。此时老和尚只须这样做：

- 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。

第2个和尚又想：如果有人能将62个盘子从一个座移到另一座，我就能将63个盘子从A座移到B座，他是这样做的：

- ① 命令第3个和尚将62个盘子从A座移到C座；
- ② 自己将1个盘子从A座移到B座；
- ③ 再命令第3个和尚将62个盘子从C座移到B座。

.....





解题思路：

为便于理解，先分析将A座上3个盘子移到C座上的过程：

- ① 将A座上2个盘子移到B座上（借助C座）。
- ② 将A座上1个盘子移到C座上。
- ③ 将B座上2个盘子移到C座上（借助A座）。

其中第②步可以直接实现。第①步又可用递归方法分解为：

将A座上1个盘子从A座移到C座；

将A座上1个盘子从A座移到B座；

将C座上1个盘子从C座移到B座。

第③步可以分解为：

将B座上1个盘子从B座移到A座上；

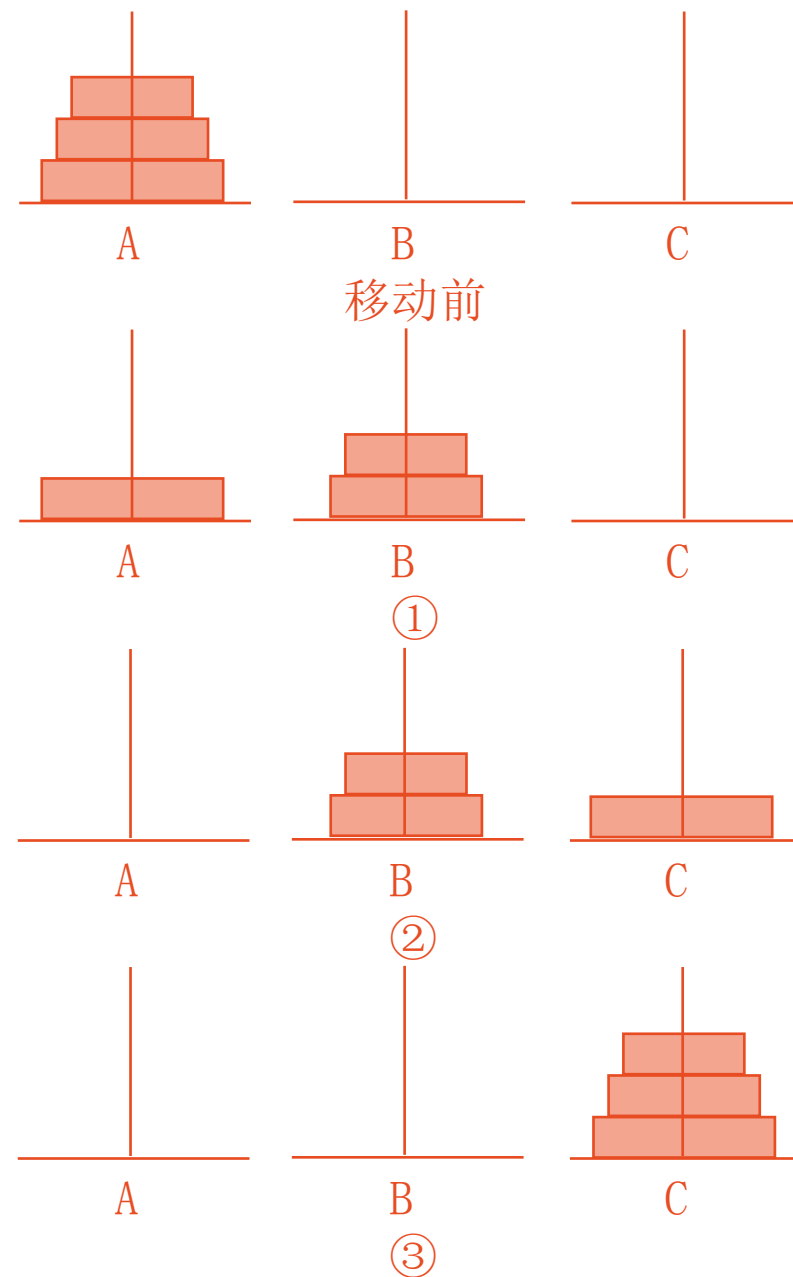
将B座上1个盘子从B座移到C座上；

将A座上1个盘子从A座移到C座上。

将以上综合起来，可得到移动3个盘子的步骤为：

$A \rightarrow C$, $A \rightarrow B$, $C \rightarrow B$, $A \rightarrow C$, $B \rightarrow A$, $B \rightarrow C$, $A \rightarrow C$ 。

共经历7步。由此可推出：移动n个盘子要经历 $(2^n - 1)$ 步。





解题思路：

由上面的分析可知：将 n 个盘子从A座移到C座可以分解为以下3个步骤：

- ① 将A座上 $n-1$ 个盘借助C座先移到B座上；
- ② 把A座上剩下的一个盘移到C座上；
- ③ 将 $n-1$ 个盘从B座借助于A座移到C座上。

上面第①步和第③步，都是把 $n-1$ 个盘从一个座移到另一个座上，采取的办法是一样的，只是座的字不同而已。为使之一般化，可以将第①步和第③步表示为：

将one座上 $n-1$ 个盘移到two座(借助three座)。只是在第①步和第③步中，one, two, three和A, B, C的对应关系不同。对第①步，对应关系是one对应A，two对应B，three对应C。对第③步，是：one对应B，two对应C，three对应A。

因此，可以把上面3个步骤分成两类操作：

- ① 将 $n-1$ 个盘从一个座移到另一个座上 ($n > 1$)。这就是大和尚让小和尚做的工作，它是一个递归的过程，即和尚将任务层层下放，直到第64个和尚为止。——hanoi函数
- ② 将1个盘子从一个座上移到另一座上。这是大和尚自己做的工作。——move函数



```
#include <stdio.h>
int main()
{
    void hanoi(int n, char one, char two, char three);
    //对hanoi函数的声明
    int m;
    printf("input the number of diskess:");
    scanf("%d", &m);
    printf("The step to move %d diskess:\n", m);
    hanoi(m, 'A', 'B', 'C');
}
```

```
void hanoi(int n, char one, char two, char three)    //定义
hanoi函数
//将n个盘从one座借助two座, 移到three座
{
    void move(char x, char y);    //对move函数的声明
    if(n==1)
        move(one, three);
    else
    {
        hanoi(n-1, one, three, two);
        move(one, three);
        hanoi(n-1, two, one, three);
    }
}

void move(char x, char y)    //定义move函数
{
    printf("%c->%c\n", x, y);
}
```



在本程序中，调用递归函数hanoi，其终止条件为hanoi函数的参数n的值等于1。显然，此时不必再调用hanoi函数了，直接执行move函数即可。在本程序中move函数并未真正移动盘子，而只是输出移盘的方案（表示从一个座移到哪一个座）。

```
C:\WINDOWS\system32\cmd.exe
input the number of diskess:3
The step to move 3 diskess:
A->C
A->B
C->B
A->C
B->A
B->C
A->C
请按任意键继续. . .
```

数组作为函数参数

形式参数	实际参数
变量	常量、变量、表达式、数组元素
数组	数组

定义：

```
int max1(int a, int b)
{
}
int max2(int a[], int n)
{
}
```

调用：

```
int x, y, c[100], m;
m = max1(x, y);
m = max2(c, 100);
```

实参x, y值copy至形参a, b。值传递！

实参c至形参a是如何传递的？

实参数组c首元素地址copy至形参a。

数组元素作为函数实参

数组元素可以用作函数实参，但是不能用作形参。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存储单元(数组是一个整体，在内存中占连续的一段存储单元)。在用数组元素作函数实参时，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。

定义：

```
int max1(int a, int b)
{
}
int max2(int a[10], int n)
{
}
```

调用：

```
int x[10], m;
m = max1(x[0], y[1]);
```

```
int max2(int a[], int n)
```


数组元素作函数实参

【例7.9】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。

```
#include <stdio.h>
int main()
{
    int max(int x,int y);           //函数声明
    int a[10],m,n,i;
    printf("enter 10 integer numbers:");
    for(i=0;i<10;i++)               //输入10个数给a[0]~a[9]
        scanf("%d",&a[i]);
    printf("\n");
    for(i=1,m=a[0],n=0;i<10;i++)
    {
        if(max(m,a[i])>m)           //若max函数返回的值大于m
        {
            m=max(m,a[i]);         //max函数返回的值取代m原值
```

```
        n=i;                       //把此数组元素的序号记下来，放在n中
    }
    printf("The largest number is %d\nit is the %dth
number.\n",m,n+1);
}

int max(int x,int y)               //定义max函数
{
    return(x>y?x:y);               //返回x和y中的大者
}
```



从键盘输入10个数给a[0]~a[9]。变量m用来存放当前已比较过的各数中的最大者。开始时设m的值为a[0]，然后依次将m与a[i]比，如果a[i]大于m，就以a[i]的值取代m的原值。下一次以m的新值与下一个a[i]比较。经过9轮循环的比较，m最后的值就是10个数的最大数。

请注意分析怎样得到最大数是10个数中第几个数。当每次出现以max(m, a[i])的值取代m的原值时，就把i的值保存在变量n中。n最后的值就是最大数的序号(注意序号从0开始)，如果要输出“最大数是10个数中第几个数”，应为n+1。因为数组元素序号从0开始。

```
C:\WINDOWS\system32\cmd.exe
enter 10 integer numbers:4 7 0 -3 4 34 67 -42 31 -76
The largest number is 67
it is the 7th number.
请按任意键继续. . .
```

一维数组名作函数参数

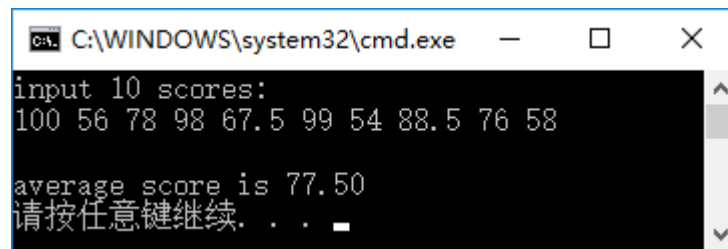
注意

- 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7.10】有一个一维数组score，内放10个学生成绩，求平均成绩。

```
#include <stdio.h>
int main()
{
    float average(float array[10]); //函数声明
    float score[10], aver;
    int i;
    printf("input 10 scores:\n");
    for(i=0; i<10; i++)
        scanf("%f", &score[i]);
    printf("\n");
    aver=average(score); //调用average函数
    printf("average score is %5.2f\n", aver);
    return 0;
}
```

```
float average(float array[]) // 替换下行
float average(float array[10]) //定义average函数
{
    int i;
    float aver, sum=array[0];
    for(i=1; i<10; i++)
        sum=sum+array[i]; //累加学生成绩
    aver=sum/10;
    return(aver);
}
```



```
C:\WINDOWS\system32\cmd.exe
input 10 scores:
100 56 78 98 67.5 99 54 88.5 76 58

average score is 77.50
请按任意键继续...
```



- (1) 用数组名作函数参数，应该在主调函数和被调用函数分别定义数组。
- (2) 实参数组与形参数组类型必须一致。
- (3) 在定义average函数时，声明形参数组的大小为10，但在实际上，指定其大小是不起任何作用的，因为C语言编译系统并不检查形参数组大小，只是将实参数组的首元素的地址传给形参数组名。
- (4) 形参数组可以不指定大小，在定义数组时在数组名后面跟一个空的方括号。

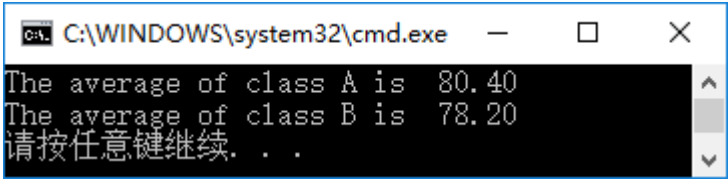
一维数组名作函数参数

【例7.11】有两个班级，分别有35和30名学生，调用average函数，分别求这两个班的学生的平均成绩。

```
#include <stdio.h>

int main()
{
    float average(float array[], int n);
    float score1[5]={98.5, 97, 91.5, 60, 55}; //定义长度为5的数组
    float
score2[10]={67.5, 89.5, 99, 69.5, 77, 89.5, 76.5, 54, 60, 99.5};
    //定义长度为10的数组
    printf("The average of class A is
%6.2f\n", average(score1, 5));
    //用数组名score1和5作实参
    printf("The average of class B is
%6.2f\n", average(score2, 10));
    //用数组名score2和10作实参
    return 0;
}
```

```
float average(float array[], int n) //定义average函数，不需指定形参
数组长度
{
    int i;
    float aver, sum=array[0];
    for(i=1; i<n; i++)
        sum=sum+array[i]; //累加n个学生成绩
    aver=sum/n;
    return(aver);
}
```



注意

- 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，这样两个数组就共占同一段内存单元。

起始地址1000	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
	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]

一维数组名作函数参数



【例7.12】用选择法对数组中10个整数按由小到大排序。

解题思路：

所谓选择法就是先将10个数中最小的数与 $a[0]$ 对换；再将 $a[1] \sim a[9]$ 中最小的数与 $a[1]$ 对换……每比较一轮，找出一个未经排序的数中最小的一个。共比较 $9(n-1)$ 轮。

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$
3	6	1	9	4
1	6	3	9	4
1	3	6	9	4
1	3	4	9	6
1	3	4	6	9


未排序时的情况

将5个数中最小的数1与 $a[0]$ 对换

将余下的后面4个数中最小的数3与 $a[1]$ 对换

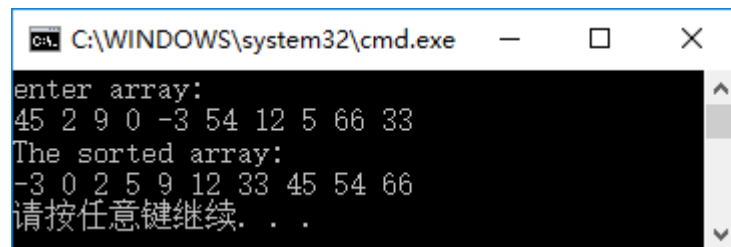
将余下的3个数中最小的数4与 $a[2]$ 对换

将余下的2个数中最小的数6与 $a[3]$ 对换，至此完成排序



```
#include <stdio.h>
int main()
{
    void sort(int array[],int n);
    int a[10],i;
    printf("enter array:\n");
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    sort(a,10); //调用sort函数,a为数组名,大小为10
    printf("The sorted array:\n");
    for(i=0;i<10;i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

```
void sort(int array[],int n)
{
    int i,j,k,t;
    for(i=0;i<n-1;i++)
    {
        k=i;
        for(j=i+1;j<n;j++) // 未经排序的数从i+1开始,
            if(array[j]<array[k])
                k=j; // 未经排序的数中最小数的下标
        t=array[k]; array[k]=array[i]; array[i]=t;
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
enter array:
45 2 9 0 -3 54 12 5 66 33
The sorted array:
-3 0 2 5 9 12 33 45 54 66
请按任意键继续. . .
```



可以看到在执行函数调用语句“sort(a,10);”之前和之后，a数组中各元素的值是不同的。原来是无序的，执行“sort(a,10);”后，a数组已经排好序了，这是由于形参数组array已用选择法进行排序了，形参数组改变也使实参数组随之改变。

用数组元素作实参时，向形参变量传递的是数组元素的值，称为值传递；而用数组名作函数实参时，向形参（数组名或指针变量）传递的是数组首元素的地址，称为“地址传递”，形参数组的改变也是实参数组随之改变。

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。



```
int array[3][10]; 或 int array[][10];    //二者等价
```



```
int array[][]; 或 int array[3][ ];    //必须指定列数
```

在定义二维数组时，**必须指定列数**(即一行中包含几个元素)，由于形数组与实数组类型相同，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(行数)而省略第2维(列数)。

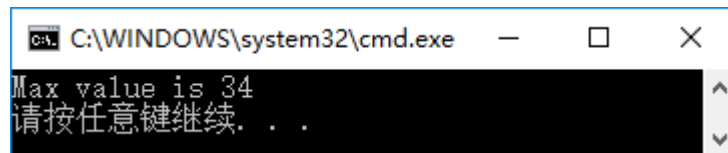
在第2维大小相同的前提下，形数组的第1维可以与实数组不同。例如，实数组定义为`int score[5][10]`；而形数组定义为`int array[][10]`；或`int array[8][10]`；均可以。这时形数组和实数组都是由相同类型和大小的一维数组组成的。C语言编译系统不检查第一维的大小。

多维数组名作函数参数

【例7.13】有一个3×4的矩阵，求所有元素中的最大值。

```
#include <stdio.h>
int main()
{
    int max_value(int array[][4]);           //函数声明
    int a[3][4]={ {1, 3, 5, 7}, {2, 4, 6, 8}, {15, 17, 34, 12}}; //对数
    组元素赋初值
    printf("Max value is %d\n", max_value(a));
    //max_value(a)为函数调用
    return 0;
}
```

```
int max_value(int array[][4])    //函数定义
{
    int i, j, max;
    max=array[0][0];
    for(i=0;i<3;i++)
        for(j=0;j<4;j++)
            if(array[i][j]>max) max=array[i][j]; //把大者放在
    max中
    return(max);
}
```



形参数组array第1维的大小省略，第2维大小不能省略，而且要和实参数组a的第2维的大小相同。在主函数调用max_value函数时，把实参二维数组a的第1行的起始地址传递给形参数组array，因此array数组第1行的起始地址与a数组的第1行的起始地址相同。由于两个数组的列数相同，因此array数组第2行的起始地址与a数组的第2行的起始地址相同。a[i][j]与array[i][j]同占一个存储单元，它们具有同一个值。实际上，**array[i][j]就是a[i][j]**，在函数中对array[i][j]的操作就是对a[i][j]的操作。

局部变量和全局变量

每一个变量都有一个作用域问题，即它们在什么范围内有效。

局部变量

定义变量可能有3种情况：

- (1) 在函数的开头定义；
- (2) 在函数内的复合语句内定义；
- (3) 在函数的外部定义。

在一个函数内部定义的变量只在本函数范围内有效，也就是说只有在本函数内才能引用它们，在此函数以外是不能使用这些变量的。在复合语句内定义的变量只在本复合语句范围内有效，只有在本复合语句内才能引用它们。在该复合语句以外是不能使用这些变量的，以上这些称为“**局部变量**”。

局部变量

```
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
```

(1) 主函数中定义的变量也只在主函数中有效。主函数也不能使用其他函数中定义的变量。

(2) 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。

(3) 形式参数也是局部变量。只在定义它的函数中有效。其他函数中不能直接引用形参。

(4) 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，这种复合语句也称为“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
```

全局变量（外部变量）

程序的编译单位是源程序文件, 一个源文件可以包含一个或若干个函数。在函数内定义的变量是局部变量, 而在函数之外定义的变量称为**外部变量**, 外部变量是**全局变量** (也称全程变量)。全局变量可以为本文件中其他函数所共用。它的有效范围为从定义变量的位置开始到本源文件结束。

注意

- 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。

全局变量（外部变量）

```
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

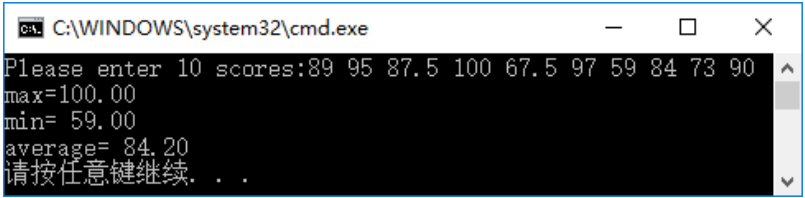
全局变量c1, c2的作用范围

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。

*为了便于区别全局变量和局部变量，在C程序设计人员中有一个习惯（但非规定），将全局变量名的第1个字母用大写表示。

全局变量(外部变量)

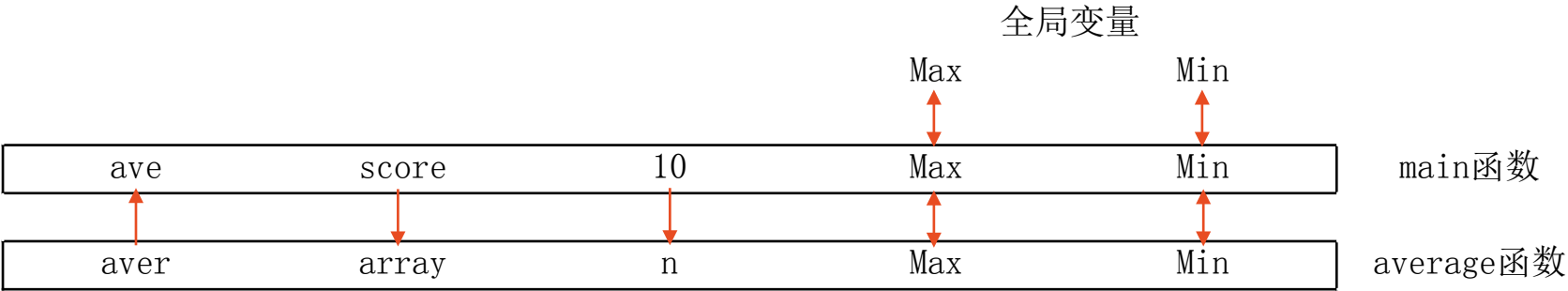
【例7.14】有一个一维数组，内放10个学生成绩，写一个函数，当主函数调用此函数后，能求出平均分、最高分和最低分。



```
#include <stdio.h>
float Max=0,Min=0; //定义全局变量Max,Min
int main()
{
    float average(float array[],int n);
    float ave,score[10];
    int i;
    printf("Please enter 10 scores:");
    for(i=0;i<10;i++)
        scanf("%f",&score[i]);
    ave=average(score,10);
    printf("max=%6.2f\nmin=%6.2f\naverage=%6.2f\n",Max,Min,ave);
    return 0;
}
```

```
float average(float array[],int n) //定义函数，有一形参是数组
{
    int i;
    float aver,sum=array[0];
    Max=Min=array[0];
    for(i=1;i<n;i++)
    {
        if(array[i]>Max) Max=array[i];
        else if(array[i]<Min) Min=array[i];
        sum=sum+array[i];
    }
    aver=sum/n;
    return(aver);
}
```

变量的关系:



全局变量（外部变量）

但是，建议不在必要时不要使用全局变量，原因如下：

- ① 全局变量在程序的全部执行过程中都**占用存储单元**，而不是仅在需要时才开辟单元。
- ② 它使函数的**通用性降低**了，因为如果在函数中引用了全局变量，那么执行情况会受到有关的外部变量的影响，如果将一个函数移到另一个文件中，还要考虑把有关的外部变量及其值一起移过去。但是若该外部变量与其他文件的变量同名时，就会出现**问题**。这就降低了程序的可靠性和通用性。在程序设计中，在划分模块时要求模块的“**内聚性**”强、与其他模块的“**耦合性**”弱。即模块的功能要单一（不要把许多互不相干的功能放到一个模块中），与其他模块的相互影响要尽量少，而用全局变量是不符合这个原则的。一般要求把C程序中的**函数做成一个相对的封闭体**，除了可以通过“**实参—形参**”的渠道与外界发生联系外，没有其他渠道。这样的程序移植性好，可读性强。
- ③ 使用全局变量过多，会降低程序的清晰性，人们往往难以清楚地判断出每个瞬时各个外部变量的值。由于在各个函数执行时都可能改变外部变量的值，程序容易出错。因此，要限制使用全局变量。

全局变量(外部变量)

【例7.15】若外部变量与局部变量同名，分析结果。

```
#include <stdio.h>
int a=3,b=5;           //a,b是全局变量
int main()
{
    int max(int a,int b); //函数声明。a,b是形参
    int a=8;             //a是局部变量
    printf("max=%d\n",max(a,b));
    return 0;
}

int max(int a,int b)    //a,b是函数形参
{
    int c;
    c=a>b?a:b;         //把a和b中的大者存放在c中
    return(c);
}
```



程序第2行定义了全局变量a和b，并对其初始化。

第3行是main函数，在main函数中(第6行)定义了一个局部变量a。局部变量a的作用范围为第6~8行。在此范围内全局变量a被局部变量a屏蔽，相当于全局变量a在此范围内不存在(即它不起作用)，而全局变量b在此范围内有效。因此第6行中max(a,b)的实参a应是局部变量a，所以max(a,b)相当于max(8,5)。它的值为8。

第10行起定义max函数，形参a和b是局部变量。全局变量a和b在max函数范围内不起作用，所以函数max中的a和b不是全局变量a和b，而是形参a和b，它们的值是由实参传给形参的，即8和5。

若外部变量与局部变量同名，局部变量屏蔽全局变量，即局部变量优先使用。

静态局部变量(static局部变量)

【例7.16】考察静态局部变量的值。

```
#include <stdio.h>
int main()
{
    int f(int);           //函数声明
    int a=2, i;           //自动局部变量
    for(i=0; i<3; i++)
        printf("%d\n", f(a)); //输出f(a)的值
    return 0;
}

int f(int a)
{
    auto int b=0;         //自动局部变量, int b=0;
    static int c=3;       //静态局部变量
    b=b+1;
    c=c+1;
    return(a+b+c);
}
```

静态局部变量:

函数被第1次调用时, 仅初始化一次。
以后的值是上次函数调用时的值。



静态变量与自动变量的值的比较分析

第几次 调用	调用时初值		调用结束时的值		
	b	c	b	c	a+b+c
第1次	0	3	1	4	7
第2次	0	4	1	5	8
第3次	0	5	1	6	9

```
C:\WINDOWS\system32\cmd.exe
7
8
9
请按任意键继续. . .
```

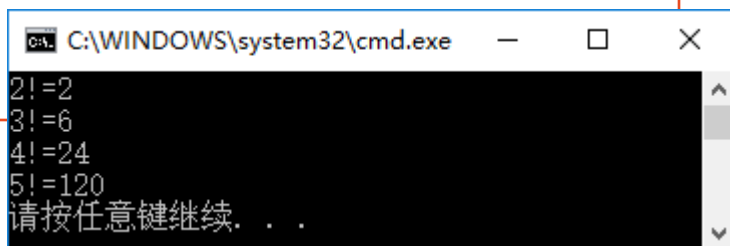

静态局部变量(static局部变量)

- (1) 静态局部变量属于静态存储类别，在静态存储区内分配存储单元。在程序整个运行期间都不释放。而自动变量（即动态局部变量）属于动态存储类别，分配在动态存储区空间而不在静态存储区空间，函数调用结束后即释放。
- (2) 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再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。
- (3) 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的值。这是由于每次函数调用结束后存储单元已释放，下次调用时又重新另分配存储单元，而所分配的单元中的内容是不可知的。
- (4) 虽然静态局部变量在函数调用结束后仍然存在，但其他函数是不能引用它的。因为它是局部变量，只能被本函数引用，而不能被其他函数引用。

静态局部变量

【例7.17】输出1到5的阶乘值。

```
#include <stdio.h>
int main()
{
    int fac(int n);
    int i;
    for(i=1;i<=5;i++)    //先后5次调用fac函数
        printf("%d!=%d\n", i, fac(i)); //每次计算并输出i!的值
    return 0;
}
int fac(int n)
{
    static int f=1;    //f保留了上次调用结束时的值
    f=f*n;              //在上次的f值的基础上再乘以n
    return(f);         //返回值f是n!的值
}
```



```
C:\WINDOWS\system32\cmd.exe
2!=2
3!=6
4!=24
5!=120
请按任意键继续. . .
```



(1) 每次调用fac(i)，输出一个i!，同时保留这个i!的值以便下次再乘(i+1)。

(2) 如果函数中的变量只被引用而不改变值，则定义为静态局部变量(同时初始化)比较方便，以免每次调用时重新赋值。

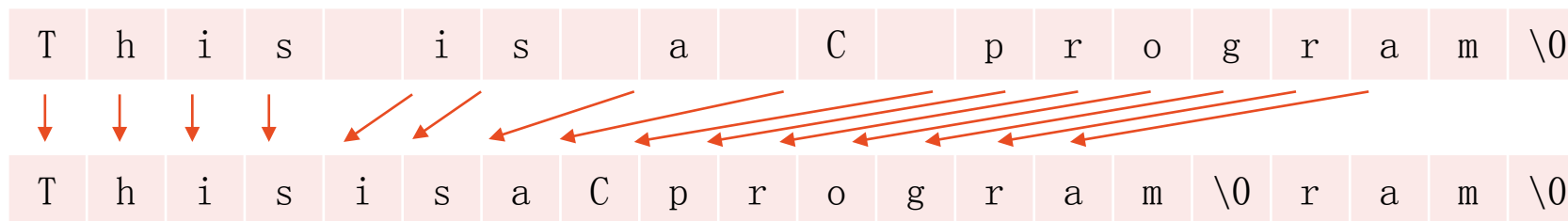
注意

- 用静态存储要多占内存（长期占用不释放，而不能像动态存储那样一个存储单元可以先后为多个变量使用，节约内存），而且降低了程序的可读性，当调用次数多时往往弄不清静态局部变量的当前值是什么。因此，若非必要，不要多用静态局部变量。

函数应用: 程序设计模块化

【例7.20】有一个字符串,内有若干个字符,现输入一个字符,要求程序将字符串中该字符删去。

解题思路: 设要删除空格

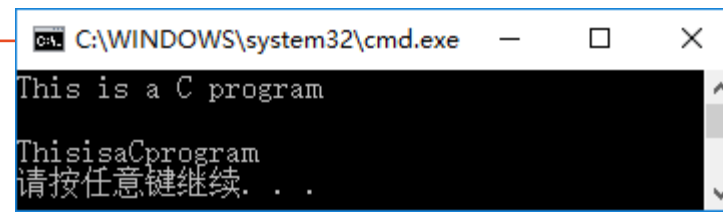


```
#include <stdio.h>
int main()
{
    void enter_string(char str[]);           //对函数的声明
    void delete_string(char str[],char ch); //对函数的声明
    void print_string(char str[]);          //对函数的声明
    char c, str[80];
    enter_string(str); //调用enter_string函数,输入字符串
    scanf("%c",&c);    //输入要求删去的字符
    delete_string(str,c); //调用delete_string函数,删除str中的c
    print_string(str);  // 调用print_string函数,输出
    return 0;
}
```

```
void enter_string(char str[80]) //定义外部函数enter_string
{
    gets(str);                  //向字符数组输入字符串
}
```

```
void delete_string(char str[],char ch)
{
    int i,j;
    for(i=j=0;str[i]!='\0';i++)
        if(str[i]!=ch)
            str[j++]=str[i];
    str[j]='\0';
}
```

```
void print_string(char str[])
{
    printf("%s\n",str);
}
```



输入一个9位以内的正整数n，按数值从高到低的顺序输出n的各位数字。

```
int main()
{
    int i = 0, j, k, n, num[9], t;
    scanf("%d", &n);
    while(n) {
        num[i++] = n%10;
        n /= 10;
    }
    k = i--; // k: 数字个数

    // 冒泡排序
    for(j = 1; j <= k-1; j++)
        for(i = 0; i < k - j; i++)
            if (num[i] < num[i+1])
                { t = num[i]; num[i] = num[i+1]; num[i+1] = t; }

    // 输出
    for(j = 0; j < k; j++) printf("%d ", num[j]);
    printf("\n");
    return 0;
}
```

输入一个9位以内的正整数n，按数值从高到低的顺序输出n的各位数字。

```
int main()
{
    void order(int a[],int n); // 排序函数说明

    int i = 0, j, k, n, num[9];
    scanf("%d", &n);
    while(n) {
        num[i++] = n%10;
        n /= 10;
    }
    k = i--; // k: 数字个数

    order(num, k); // 排序函数说明

    // 输出
    for(j = 0; j < k; j++) printf("%d ", num[j]);
    printf("\n");
    return 0;
}
```

```
// 排序函数定义
void order(int a[],int n)
{
    int i, j, t;
    // 冒泡排序
    for(j = 1; j <= n-1; j++)
        for(i = 0; i < n - j; i++)
            if (a[i] < a[i+1])
            {
                t = a[i];
                a[i] = a[i+1];
                a[i+1] = t;
            }
}
```

小结

- 理解使用函数进行模块化程序设计的思想

- 函数说明；函数定义；函数调用

- 形式参数和实际参数, 返回值

```
int fun(int x){ }; int a = 10, b; b = fun(a);
```

- 函数参数传递：值传递；地址传递

```
void fun(int x); void fun(int a[]);
```

- 函数的递归调用

递归结束条件，有返回值的递归，递归调用自己不要忘记return语句。

```
int fun(int n) { ... return(fun(n-1)); }
```

- 局部变量与全局变量

函数调用后，函数内部的局部变量（动态）存储区消失。全局变量存储区在整个程序运行期间都存在

- 静态局部变量

静态局部变量存储区在整个程序运行期间都存在，第1次函数调用时，静态局部变量仅被初始化1次。