计算机程序设计

刘龙军 副教授

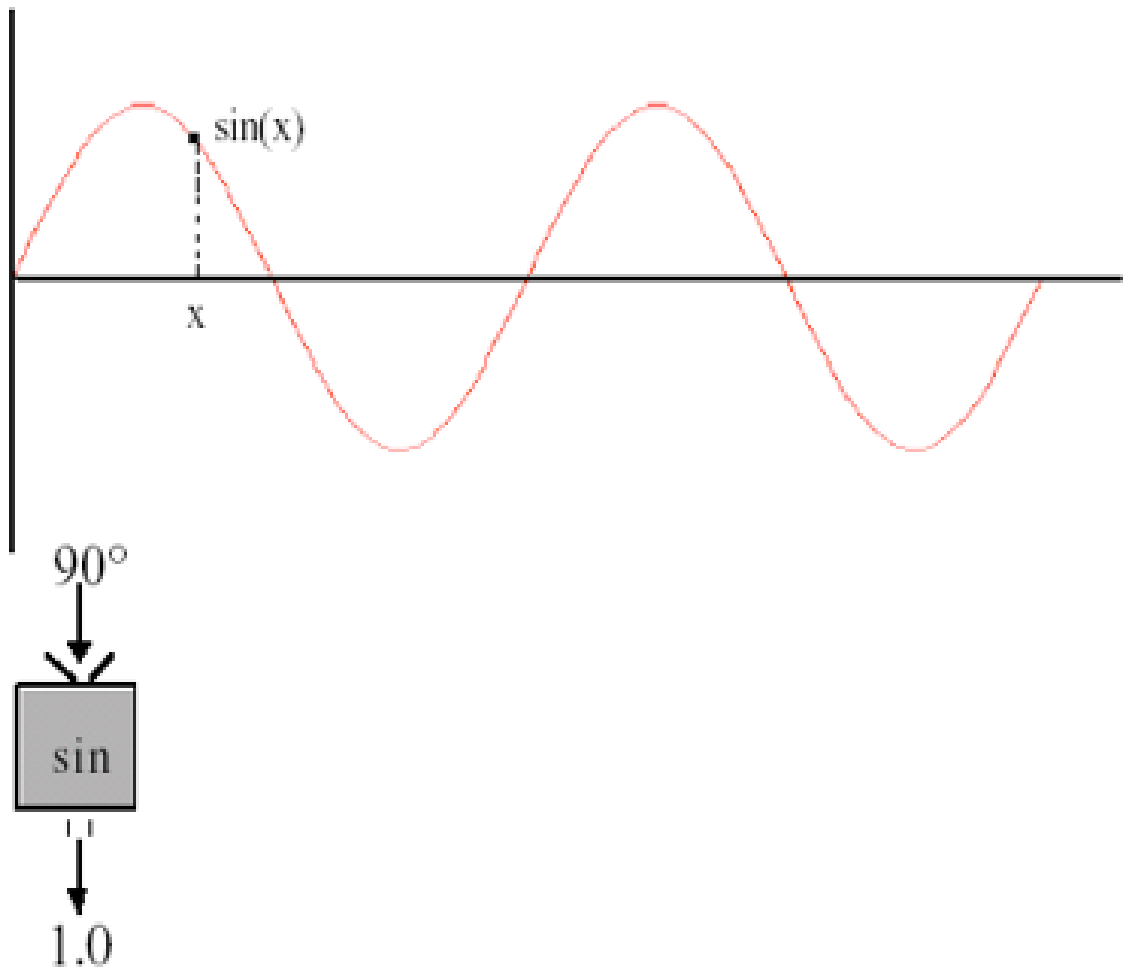
2025年9月29日

目录

函数与数组

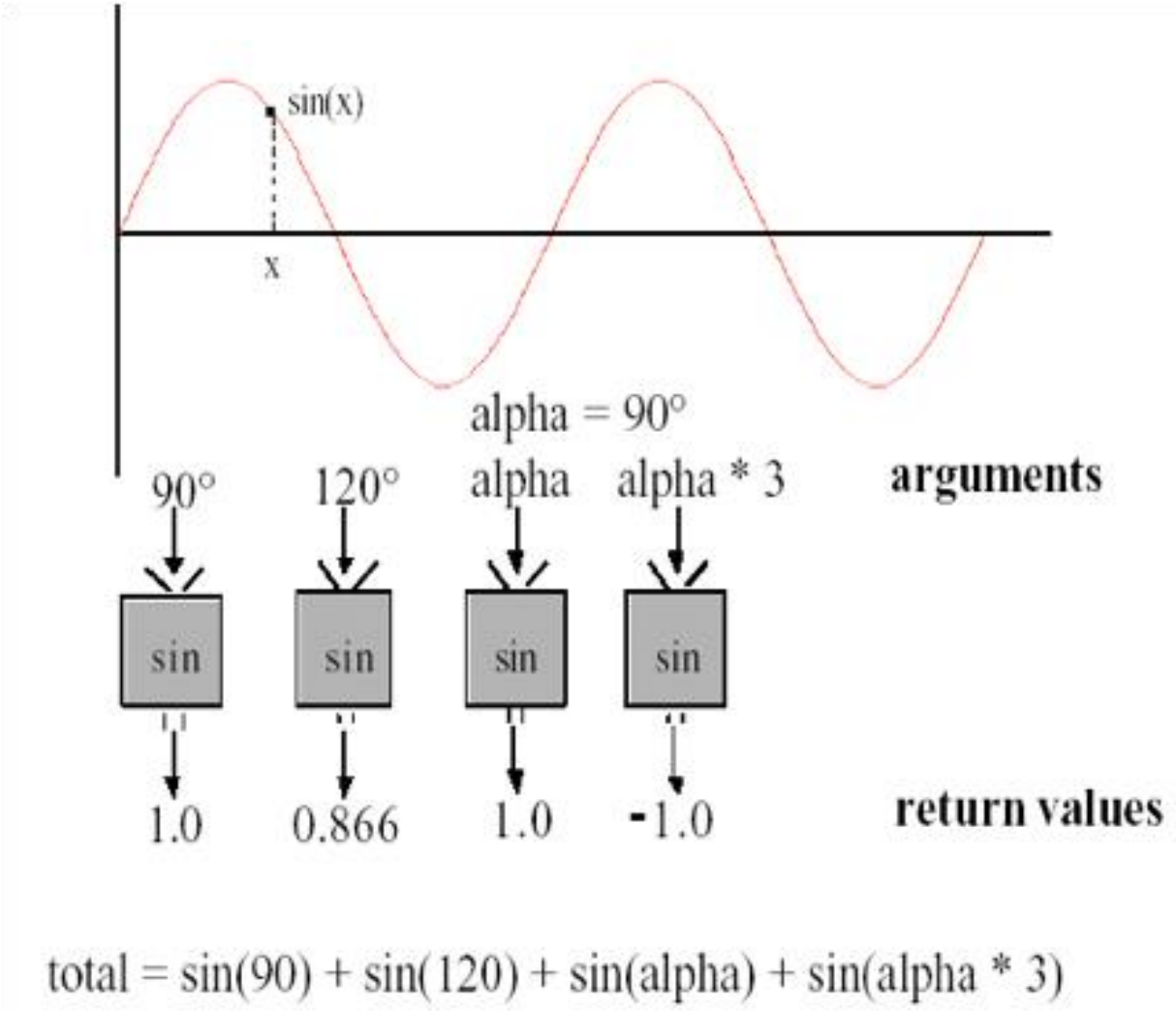
数学函数

- 我们可以将像sin(x)那样的数学函数想象成一个黑盒子，或一个小机器。如果你在它的上面放入一个“值”，在它的下面就会掉出“结果”
- 对应于程序函数，上面的值称为参数，下面的值称为返回值



调用sin(x)函数

- 如果我们改变了输入的参数，函数就能返回不同的值。
- 图中我们将调用4次sin的结果加起来，并将其和存入变量total中。



函数的本质

- 函数是这样一种运算：
  - **函数名**说明运算规则
  - **参数**是运算的操作数
  - **返回值**是运算的结果

```
int add(int a, int b,
int c)
{
    return a+b+c;
}
```

```
x = a + b + c;
```



```
x = add(a, b, c);
```

举例

- 某同学去某公司工作：第一个月和第二个月老板各给100，第三个月开始，每月的工资是前两个月工资的和。
- 该同学工作一年后的当月工资是多少？一年来总共收入多少？（如果二十年呢？）

问题的本质

- 本质是求斐波那契数列（Fibonacci sequence）

  F(N) = F(N-1) + F(N-2)

  其中，F(1) = F(2) = 100， 3<=N

- 斐波那契数列指的是这样一个数列 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233，377，610，987，1597，2584，4181，6765，10946，17711，28657，46368

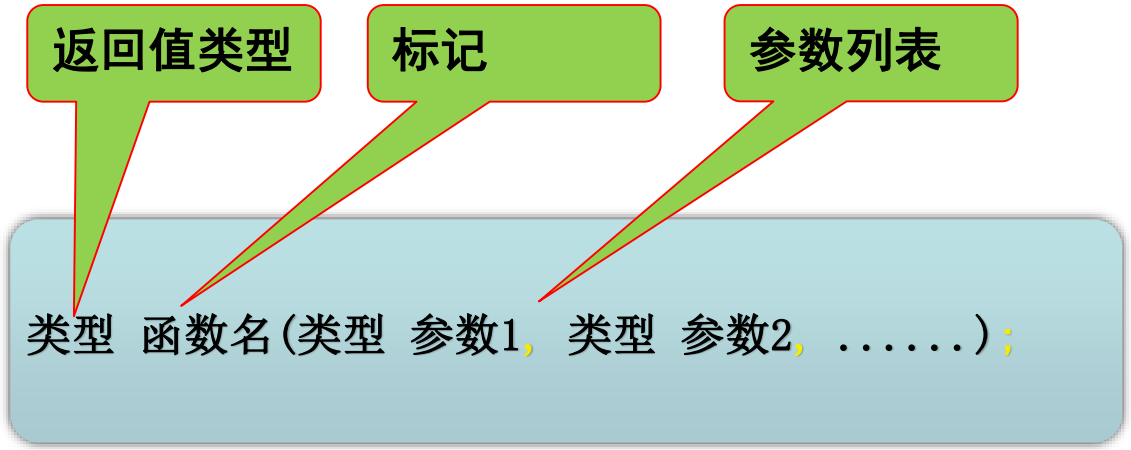
函数的作用

- **分解问题，使程序结构更加清晰**
  - 在解决复杂问题时，先**将问题划分或化解成一些子问题分别加以解决**，从而降低问题的复杂度。在程序实现时可针对这些子问题分别编程实现和测试。（**分而治之**）
- **避免重复劳动**
  - 在解决不同的实际问题时，常常需要用到**许多相同的子方法或子技术**。它们对于不同问题可能只是选择参数的不同，而内部功能完全相同。如果能将这些子方法或子技术形成一个经过测试的优化的固定程序模块，让用户在解决问题时直接调用，就会比较方便。

学习函数的三个要素–

函数的声明、定义与调用

函数声明



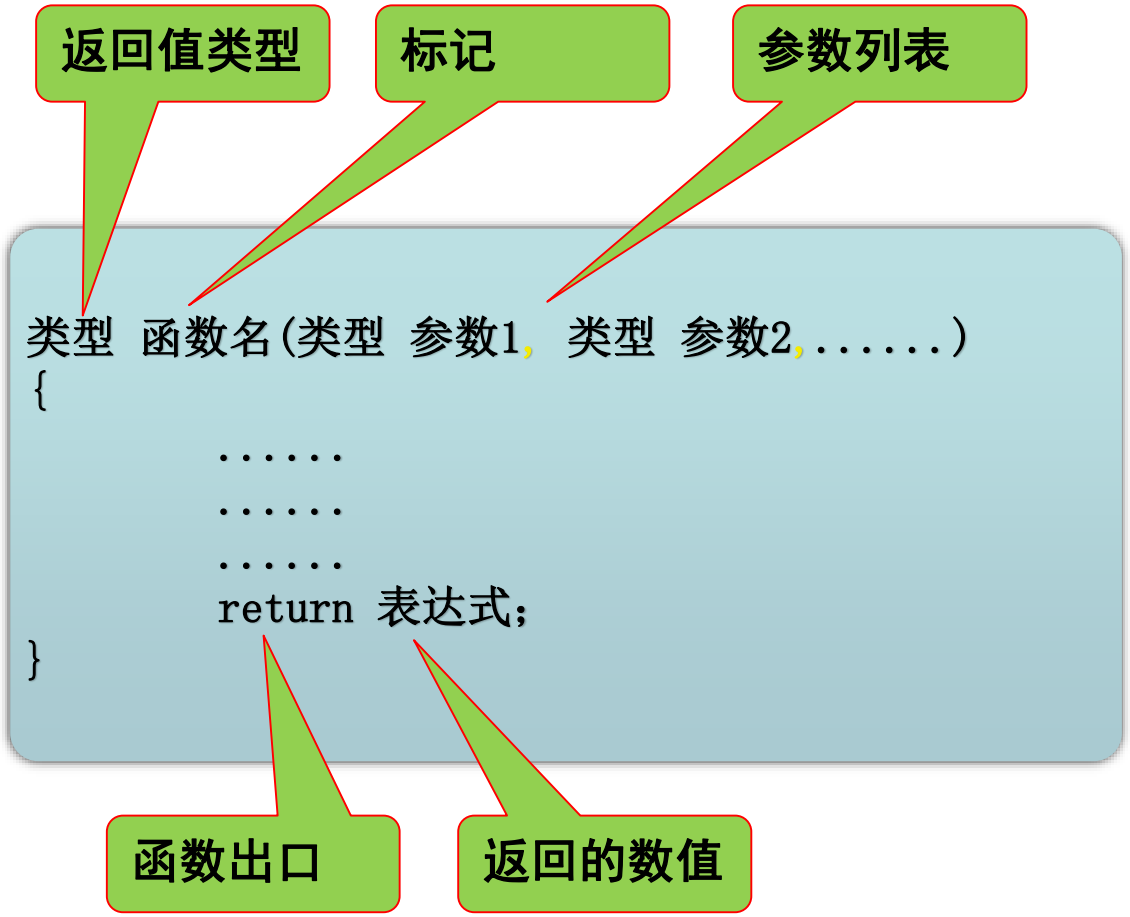
例：

```
int CalRabbitsNum(int vMonth);
int RabbitsAmount(int month);
int max(int x, int y);
void printf_message();
```

函数声明注意事项

- 在使用某个函数之前，必须知道函数在哪里，即**编译器需要看到该函数的声明或定义**
- .h 文件

函数定义




```
int max(int x, int y)
{
     int max;
    if(x>y)
      max =x;
     else
      max =y;
    return max;
}
```

函数定义的三种形式

有参函数

```
类型 函数名(类型 参数1, 类型 参数2,.....)
{
        ......
        return 表达式;
}
```

无参函数

```
类型 函数名()
{
        ......
        return 表达式;
}
```

空函数

```
void 函数名()
{
        ......
}
```

```
//max.cpp

#include <iostream>
using namespace std;

int max(int x, int y)
{
    int max;
    if(x>y)
        max =x;
    else
        max =y;
    return max;
}

int main()
{
    int a,b;
    cin>>a>>b;
    cout<<max(a,b)<<endl;

    return 0;
}
```

函数的返回值

- 函数通过return语句将函数计算结果返回
- 当函数执行到return语句 或 } 时，函数的运算停止。程序从当此调用函数的地方继续执行

  -函数可以有多个return

```
int max(int x, int y)
{
    if(x>y)
     return x;
    else
     return y;
}
```

```
int max(int x, int y)
{
    int max;
    if(x>y)
            max =x;
    else
       max =y;
    return max;
}
```

不返回任何值的函数—空函数（void）

- 有时某些函数没有返回值，调用过程的输出已经在被调函数体内完成,这时要用**void定义**这些“无返回值”类型的函数
- 此时，return语句可以省略

```
void print_star()
{
    cout<<"*******\n";
}

void print_message()
{
    cout<<"how are u?\n";
}
```

例子

```
int  max (int x,int y)
{   int  z;
    z=(x>y)? x : y ;
    return  z;
}
void main (void )
{    int   a,b,c;
     cin>>a>>b;
     c=max (a , b) ;
     cout<<"The max is "<<c<<endl;
}
```

函数类型

形参列表说明

函数名

函数体

函数值

主调函数

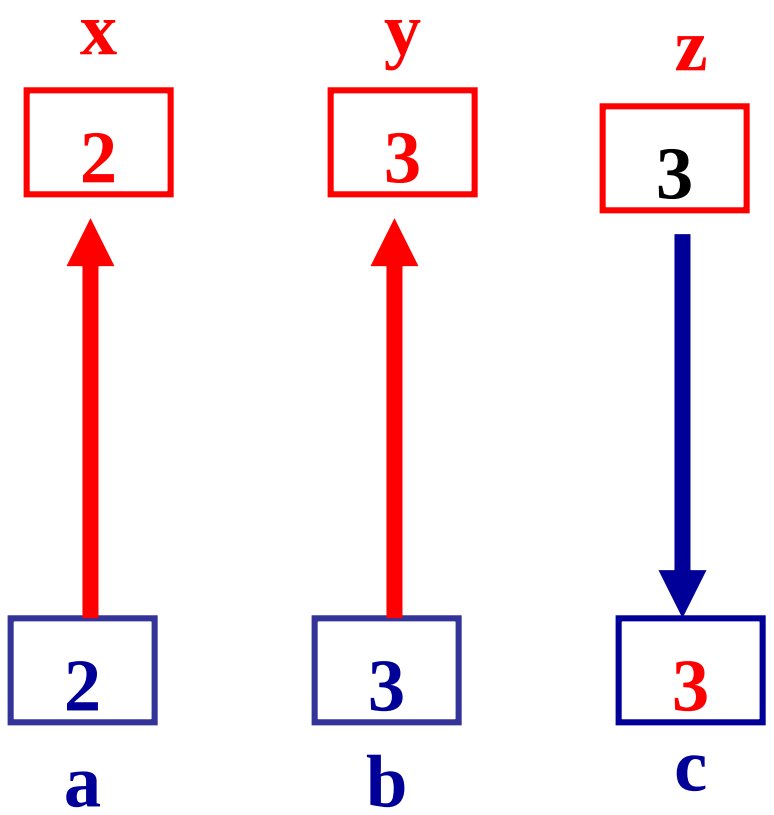
调用函数

实际参数

将实际值a,b传给被调函数的参数x,y，计算后得到函数值z返回

例子

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

 void main (void )
{    int   a,b,c;
     cin>>a>>b;
     c=max (a , b) ;
     cout<<"The max is"<< c<<endl;
}
```

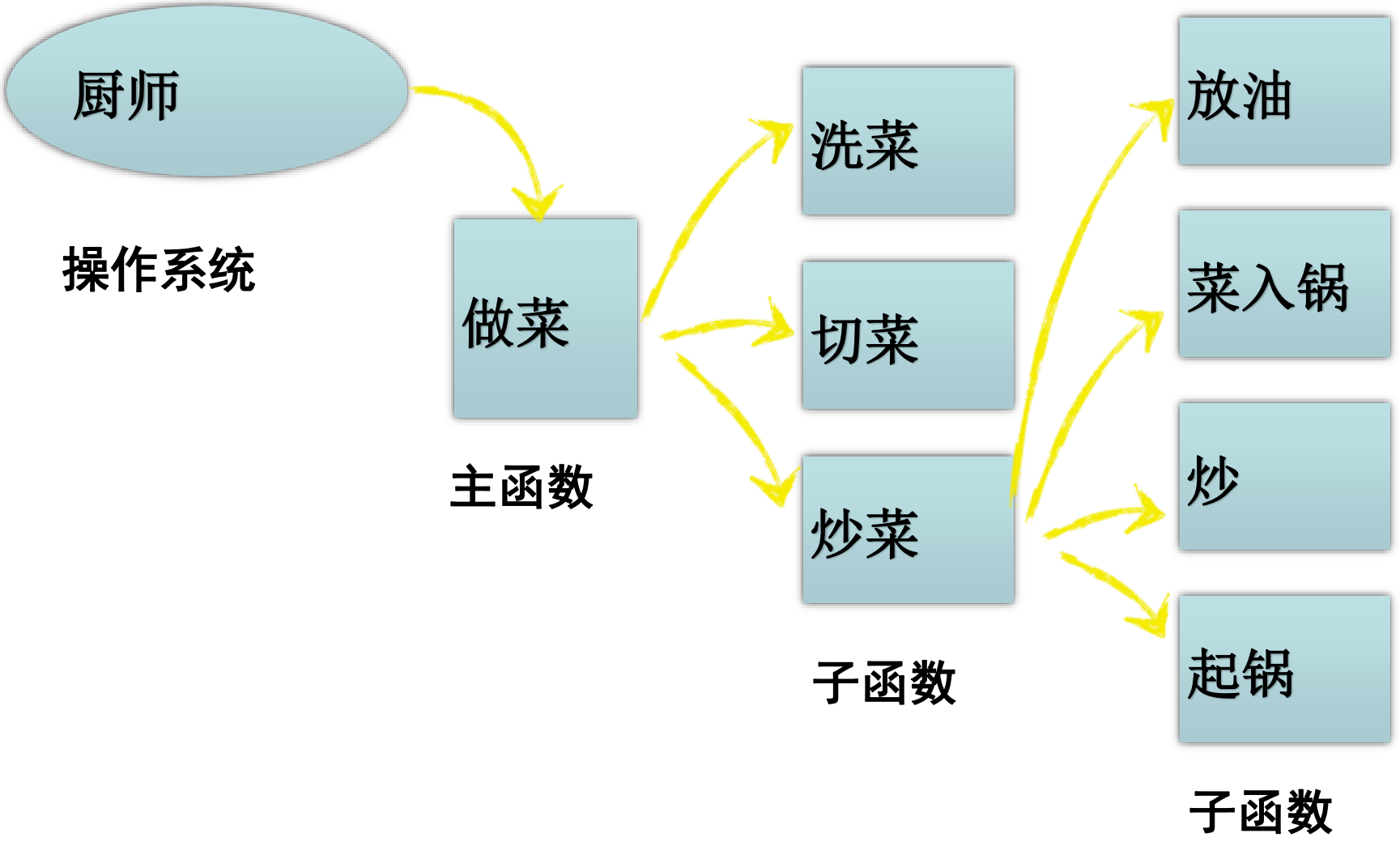


函数调用

- **函数调用的形式：**

```
函数名(参数1, 参数2, .....)
```

函数调用的示意图



函数调用的注意事项

- 调用一个函数之前，先要对其返回值类型、函数名和参数进行声明或定义
- 调用函数时，提供的参数（实际参数）和该函数定义中的参数（形式参数）必须匹配
  - 数目一致
  - 类型一一对应（会发生自动类型转换）
  - 把实际参数的值复制给形式参数（指针不同）
- 返回值可以按需处理

函数调用的方式

作为语句　　printstar( );

作为表达式　　c=max (a,b);

作为另一个函数的参数　　cout<<max (a,b);

注意

- 在一个函数中调用另一函数（即**被调用函数**）需要具备哪些条件呢？
  - 1) 被调用的函数必须是**已存在的函数**
  - 2) 如果使用**库函数**，必须用 #include < math.h>
  - 3) 函数调用遵循**先定义、后调用的原则**，即**被调函数应出现在主调函数之前。**

例子

```
float max(float x, float y)
{   float  z;
    z=(x>y)? x : y ;
    return  z;
}

void main (void)
{  float   a,b, c;
   cin>>a>>b;
   c=max (a+b , a*b) ;
   cout<<“The max is”<<c<<endl;
}
```

形参必须说明参数类型

被调函数先定义

定义之后再调用

实参传递的是一个具体的值，不必说明参数类型

函数参数传递

函数参数

- 函数参数：就是一个变量，它随着使用者的不同而发生变化。
  - 形参：函数定义中的形式参数。
  - 实参：就是在函数调用时，传递给函数的参数，也就是参数的实际值

```
int bar(int c, int d)
{
  int res = c + d;
  return res;
}

int foo(int a, int b)
{
  int res = bar(a, b);
  return res;
}

int main()
{
   foo(2, 3);
   return 0;
}
```

c,d 为形参

a,b 为形参

a,b的值为实参

2, 3 为实参

形参、实参的注意事项

- 在定义函数中指定的形参变量，在未出现函数调用时，它们并不占内存单元。只有在发生函数调用时函数中的参数才被分配内存单元。在调用结束后，形参所占的内存单元才被释放。
- 在被定义的函数中，必须指定形参的类型。
- 实参和形参类型应一致，实参可以是整型、实型、字符型或表达式，但要求它们有确定的值。

例子

```
//swap.cpp
#include <iostream>
using namespace std;\

void Swap(int x, int y);

int main()
{
    int a=123;
    int b=456;

    cout<<"a = "<<a<<"; b = "<<b<<endl;

    Swap(a,b);     //exchange a,b
    cout<<"a = "<<a<<"; b = "<<b<<endl;
    return 0;
}
void Swap(int x, int y)
{
    int tmp;
    tmp = x;
    x = y;
    y= tmp;
}
```

例子

```
//swap.cpp
#include <iostream>
using namespace std;\

void Swap(int x, int y);

int main()
{
    int a=123;
    int b=456;

    cout<<"Var\tAddress\t\tValue\n";
    cout<<"--------before swap--\n";
    cout<<"a\t"<<&a<<"\t"<<a<<endl;
    cout<<"b\t"<<&b<<"\t"<<b<<endl;

    Swap(a,b);

    cout<<"---------after swap---\n";
    cout<<"a\t"<<&a<<"\t"<<a<<endl;
    cout<<"b\t"<<&b<<"\t"<<b<<endl;
    return 0;
}
```

```
void Swap(int x, int y)
{
    int tmp;

    cout<<"---------before swap-----
--------------\n";
    cout<<"x\t"<<&x<<"\t"<<x<<endl;
    cout<<"y\t"<<&y<<"\t"<<y<<endl;

    tmp = x;
    x = y;
    y= tmp;

    cout<<"---------after swap------
-------------\n";
    cout<<"x\t"<<&x<<"\t"<<x<<endl;
    cout<<"y\t"<<&y<<"\t"<<y<<endl;

}
```

例子

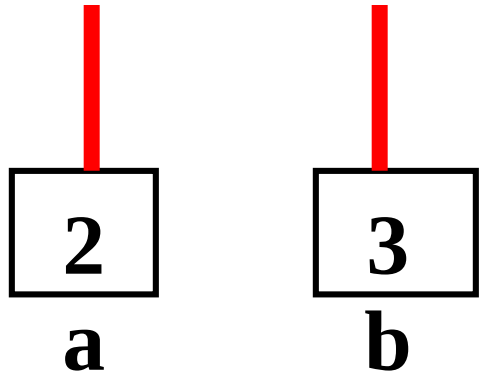
```
Var Address     Value
--------before swap--------------------
a   0x7fff5fbff7d8  123
b   0x7fff5fbff7d4  456
---------before swap--------------------
x   0x7fff5fbff73c  123
y   0x7fff5fbff738  456
---------after swap---------------------
x   0x7fff5fbff73c  456
y   0x7fff5fbff738  123
---------after swap---------------------
a   0x7fff5fbff7d8  123
b   0x7fff5fbff7d4  456
```

函数参数的传递

- 实参变量对形参变量的数据传递是“值传递”，即单向传递，只能由实参传给形参，而不能由形参传给实参，在内存中实参单元与形参单元是不同的单元。
- “地址传递”呢？

例子

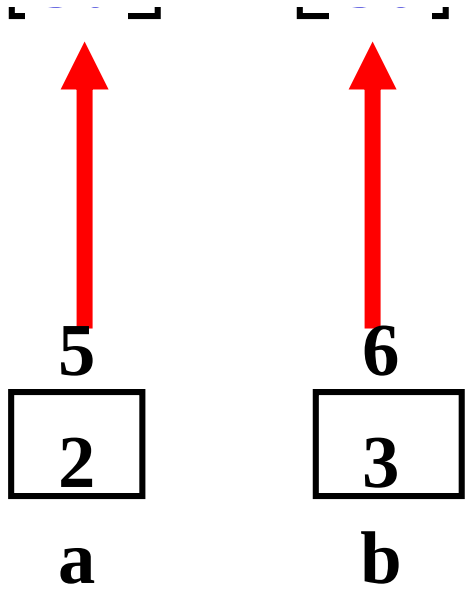
```
void fun(int a, int b)
{    a=a*10;
     b=b+a;
    cout<<a<<'\t'<<b<<endl;
}

void main(void)
{   int a=2, b=3;
    fun(a,b);
    cout<<a<<'\t'<<b<<endl;
}
```




```
20      23
2       3
```

例子

```
void fun(int x, int y)
{    x=x*10;
     y=y+x;
     cout<<x<<'\t'<<y<<endl;
}
void main(void)
{   int a=2, b=3;
    fun(a+b,a*b);
    cout<<a<<'\t'<<b<<endl;
}
```




```
50      56
2       3
```

形参实参类型相等，一一对应

注意

- 形参必须要**定义类型**，因为在定义被调函数时，不知道具体要操作什么数，而**定义的是要操作什么类型的数**。
- 形参是**被调函数中的变量**；实参是**主调函数赋给被调函数的特定值**。在函数调用语句中，**实参不必定义数据类型**，因为**实参传递的是一个具体的值(常量)**，程序可依据这个数值的表面形式来判断其类型，并将其赋值到对应的形参变量中。

数组

数组

- 程序实现中常常需要批量组织数据。例如当记录每位学生的成绩的时候，这时一般变量无法满足需求，这个时候就需要一种新的数据结构——数组。

程序设计中的数组

- 一维数组
- 二维及多维数组
- 字符数组、字符串

一维数组

声明

类型 数组名[表达式]

```
int a[10];    /* 定义包含10个整数元素的数组 */
#define NUM 10
int array[NUM];
```

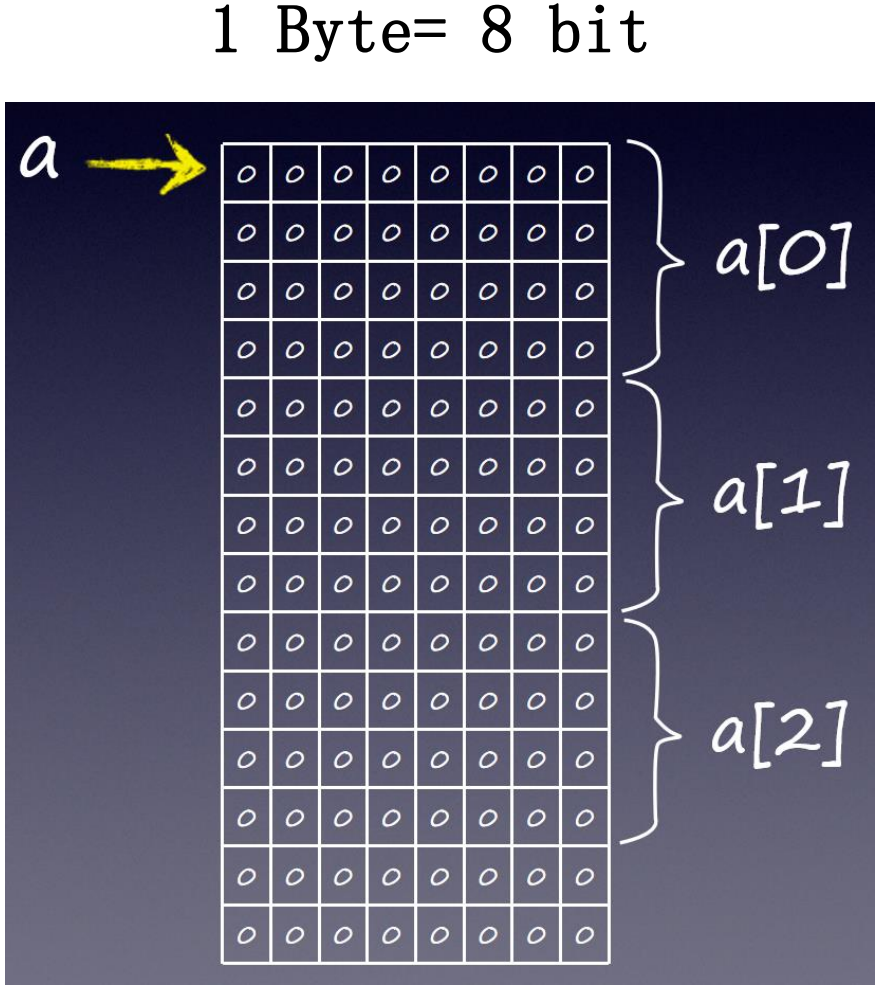
```
int count = 0;
cin>>count;
int b[count];
```

注意：

- 表达式表示数组元素的个数，下标从0开始
- VC6, VS2010不允许为变量
- gcc4, dev-cpp允许表达式为常量或变量，即支持动态数组

数组的本质

- 定义数组就是定义了一块**连续的**空间，数组元素按序存放在这块空间中。
- **数组名字是数组首元素的内存地址**。注意数组名是一个常量，不能被赋值。
- 空间的大小等于元素数*每个元素所占的空间大小，如定义int a[10]，将占用了40个字节空间，**因为每个整型数占4个字节。**



示例

```
//access.cpp
// array storage and pointer
#include <iostream>
#define NUM 10
using namespace std;
int main()
{
    int a[NUM];
    int i;

    for(i=0; i<NUM; i++){
        a[i]=i;
    }
    cout<<"Variable\tAddress\t\tContent\n";
    cout<<"--------------------------------\n";
    for(i=0; i<10; i++){
        cout<<"a["<<i<<"]\t"<<&a[i]<<"\t"<<a[i]<<"\n";
    }
    cout<<"--------------------------------\n";
    cout<<"a\t\t"<<a<<"\t"<<*a<<"\n";
    return 0;
}
```

```
Variable    Address     Content
--------------------------------
a[0]    0x7fff5fbff7c0  0
a[1]    0x7fff5fbff7c4  1
a[2]    0x7fff5fbff7c8  2
a[3]    0x7fff5fbff7cc  3
a[4]    0x7fff5fbff7d0  4
a[5]    0x7fff5fbff7d4  5
a[6]    0x7fff5fbff7d8  6
a[7]    0x7fff5fbff7dc  7
a[8]    0x7fff5fbff7e0  8
a[9]    0x7fff5fbff7e4  9
--------------------------------
a       0x7fff5fbff7c0  0
```

//注意a的意义及其地址

例子

- int a[4]; // 表明a数组由4个int型元素组成
- 其元素分别为：a[0], a[1], a[2], a[3]
- 其序号从0开始。若存放首地址为2000H，则在内存中为：

2000H	2004H	2008H	200CH	2010H
a[0]	a[1]	a[2]	a[3]	

- C++不允许对数组的大小作动态的定义，即**数组的大小不能是变量，必须是常量。**

一维数组的初始化

- 直接在定义时初始化
  - 声明数组时可以对数组初始化
  - 初始化的长度短于要被初始化的数组元素数目，那么剩余元素被初始化为0
  - 带有初始化的数组可以不定义长度，如下代码：

初始化

- 在定义数组的同时，给各数组元素赋值
- int a[10]={0,1,2,3,4,5,6,7,8,9};
- int a[10]={0,1,2,3,4};相当于

  int a[10]={0,1,2,3,4,0,0,0,0,0};
- int a[10]={0,0,0,0,0,0,0,0,0,0};相当于

  int a[10]={0};
- int a[5]={1,2,3,4,5};可写为

  int a[ ]={1,2,3,4,5};

示例

```cpp
//init.cpp
//demonstrate the initialization of the array
#include <iostream>
using namespace std;
int main()
{
    int i =0;
    int x[]={2,3,5,1,4};
    int y[5]={2,3,5};
    int z[5]={};

    cout<<"array x:\t";
    for(i=0; i<5; i++){
        cout<<x[i]<<"\t";
    }

    cout<<"\n array y:\t";
    for(i=0; i<5; i++){
        cout<<y[i]<<"\t";
    }

    cout<<"\n array z:\t";
    for(i=0; i<5; i++){
        cout<<z[i]<<"\t";
    }
    cout<<"\n";
    return 0;
}
```

```
array x:    2    3    5    1    4
 array y:   2    3    5    0    0
 array z:   0    0    0    0    0
```

数组赋值问题

- **数组的运算都是通过其元素实现的，不允许对数组进行整体赋值操作，只能使用循环逐一复制元素**

```
int a[8], b[8];
a = b;
```

- 例：将整数类型数组a和数组b的差送入数组c中，实现方法如下：

```
for(i=0; i<8; i++)
    c[i] = a[i] - b[i];
```

```
c = a - b;
```

一维数组元素访问

- 数组元素编号从 0 开始计数，元素访问格式为 x[0]、x[1]、……
- 元素的序号称为下标。程序中，下标可为整数、整型变量或结果为整型的任意表达式

数据下标越界问题

- 语言不检查数组下标的越界。如果定义数组 int a[10]，合法的下标范围是0 – 9，但如果你引用a[10]，系统不会报错，但是程序非常容易出现异常
- 解决方法：程序员在对下标变量进行操作前，仔细检查下标的合法性

数组的输入和输出

数组输入

```
    int array[3];
        int i;
for(i = 0; i < 3; i++)
{
 cin>>array[i];
}
```

```
int array[3];
    int i;
        cin>>array;
```

```
    int array[3];
        int i;
for(i = 0; i < 3; i++)
{
 cout<<array[i];
}
```

```
int array[3];
    int i;
  cout<<array;
```

一维数组应用—统计成绩

- 为了分析学生对程序设计课程的掌握程度，需要对该课程成绩进行统计。请编程序，输入一个班32名学生的程序设计课程成绩，然后按10分为一段，统计各段成绩的学生人数，并输出

“统计成绩”分析

- (1)初始化：声明并初始化成绩数组score[33] 和计数器数level[11];
- (2)输入：循环输入学生成绩到数组score;
- (3)计算：统计学生分段，并存入数组level;
- (4)输出：循环打印学生成绩分段结果;

```cpp
//score1.cpp
//instance: array
#include <iostream>
#define NUM 5
#define SCOPE 11

using namespace std;
int main()
{
    float score[NUM];
    int level[SCOPE]={};
    int i;

    for(i = 0; i < NUM; i++){
        cin>>score[i];
       int k = score[i]/10;
        level[k]++;
    }
    cout<<"Score statistics is as the following\n";
    cout<<"------------------\n";
    for(i = 0; i < SCOPE; i++){
        cout<<i*10<<" -- "<<(i+1)*10-1<<":
"<<level[i]<<"\n";
    }
    cout<<"------------------\n";

    return 0;
}
```

```
输入5个同学的成绩
67
88
83
55
99
输出如下：
成绩分段表如下:
------------------
  0 --   9:   0
 10 --  19:   0
 20 --  29:   0
 30 --  39:   0
 40 --  49:   0
 50 --  59:   1
 60 --  69:   1
 70 --  79:   0
 80 --  89:   2
 90 --  99:   1
100 -- 109:   0
------------------
```

练习

- 输入8个数，存入数组
- 逆序打印输出
- 例如：
  - 输入的8个数为：4 7 3 13 9 6 21 1
  - 输出的8个数为：1 21 6 9 13 3 7 4

计算向量内积

- 编程实现两个向量的内积计算
- 输入向量包含的元素个数n
- 输入第一个向量的元素；输入第二个向量的元素
- 输出向量的内积

定义 设有n维向量

$$x=\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix},\quad y=\begin{pmatrix}y_1\\y_2\\\vdots\\y_n\end{pmatrix},$$

令$[x,y]=x_1y_1+x_2y_2+\cdots+x_ny_n$,$[x,y]$称为向量$x$与$y$的内积.

数组作为函数参数

数组作为函数参数

- 数组作为函数参数分以下两种情形：
  - 数组元素作为函数的参数
  - 数组名作为函数的参数

数组元素作为函数的参数

- 数组元素作为函数参数跟普通变量或表达式作为函数的参数完全一样，采用值传递将数据拷贝给函数。

例子

```
void swap(int x, int y)
{
  int tmp;
  tmp = x;
  x = y;
  y = x;
}

int main()
{
  int score[NUM] = {10,20,30,40,50};
  int i=0;

  for(i=0; i< (NUM/2); i++)
  {
      swap(score[i], score[NUM-i-1]);
  }

  return 0;
}
```

数组名作为函数的参数

- 当把数组名传递给一个函数时，其实质是把该数组的首地址传递过去，即把实际参数中的数组首地址作为形式参数中的数组的首地址。

```
float average(float score[], int size);
```

- 用数组名作函数参数，实参与形参都应用数组名。这时，函数传递的是数组在内存中的地址。
- 实参中的数组地址传到形参中，实参形参共用同一段内存。

例子

```cpp
//swap_array.cpp
#include <iostream>
#include <cassert>
#define NUM 9
using namespace std;
void swap(int array[], int m, int n);
void print_array(int array[]);

int main()
{
    int score[NUM] =
{10,20,30,40,50,60,70,80,90};
    int i = 0;
   for(i=0; i< (NUM/2); i++)
    {
      swap(score, i, NUM-i-1);
    }

    print_array(score);     //输出交换后的
数组
    return 0;
}
```

```cpp
void swap(int array[], int m, int n)
{

cout<<"array["<<m<<"]:\t"<<&array[m]<<"\t"<<array[m]<<
endl;

cout<<"array["<<n<<"]:\t"<<&array[n]<<"\t"<<array[n]<<
endl;

    int tmp;
    tmp = array[m];
    array[m] = array[n];
    array[n] = tmp;
}

void print_array(int array[])
{
    int i;
    for(i=0; i<NUM; i++)
    {
        cout<<array[i]<<"\t";
    }
    cout<<endl;
}
```

例子

编程求程设课程的平均成绩

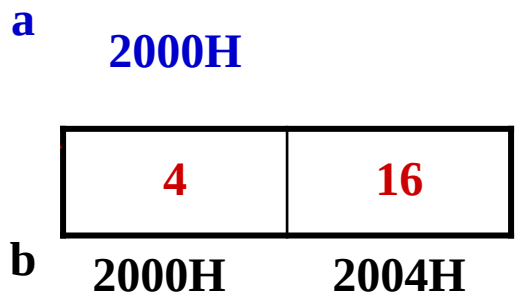
```cpp
//average.cpp
#include <iostream>
#define NUM 10
using namespace std;
float average(float array[], int length);

int main()
{
    float score[NUM] = {78,82,85,60,90,100,98,94,99,100};
    float avg= average(score, NUM);
    cout<<"Average score = "<<avg<<endl;
    return 0;
}

float average(float array[], int length)
{
    int i;
    float avg=0;
    float sum=0;

    for(i = 0; i < length; i++) {
        sum = sum + array[i];
    }
    avg = sum/length;
    return avg;
}
```

例子

```
void  fun(int a[2])
{  for(int i=0;i<2;i++)
      a[i]=a[i]*a[i];
 }
void main(void)
{   int b[2]={2,4};
    cout<<b[0]<<'\t'<<b[1]<<endl;
    fun(b);
   cout<<b[0]<<'\t'<<b[1]<<endl;
}
```

数组b和数组a占据同一段内存

a同样为数组首地址，也是2000H



b就是2000H

输出：2　4

4　16

例子

```
void  sort(int x[ ], int n)
{  int t,i,j;
   for( i=0;i<n-1;i++)
       for(j=0;j<n-i-1;j++)
          if(x[j]>x[j+1])
            {  t=x[j]; x[j]=x[j+1]; x[j+1]=t;}
  }
void main(void)
{   int a[5]={20,4,16,8,10};
     sort(a, 5 );
    for(int i=0;i<5;i++)
       cout<<a[i]<<'\t';
}
```

x

a

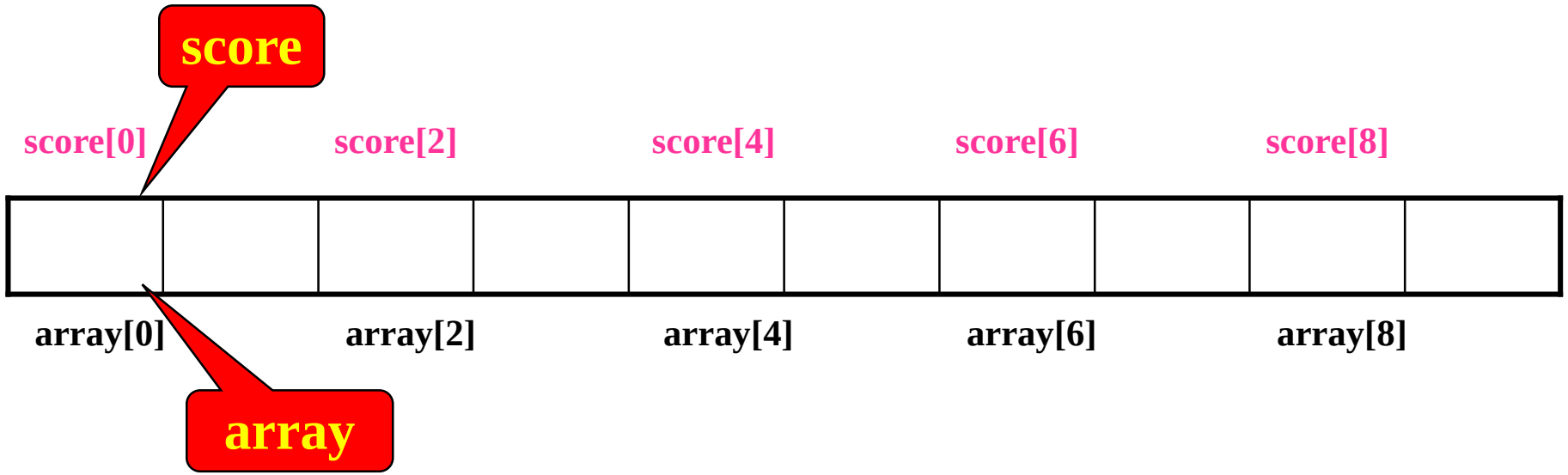
4	8	10	16	20

注意

- 1、用数组名作函数参数，应在主调函数和被调函数中分别定义数组，且类型一致。
- 2、需指定实参数组大小，形参数组的大小可不指定。数组名作实参实际上是传递数组的首地址。

注意

- 3、数组名代表数组在内存中存储的首地址，这样，数组名作函数实参，实际上传递的是数组在**内存中的首地址**。实参和形参**共占一段内存单元**，形参数组中的值发生变化，也相当于实参数组中的值发生变化。



本章结束！