# 第十讲　数组

1 一维数组的声明与引用

2 二维数组的声明及引用

3 数组作为函数参数

4 字符数组

5 对象数组

# 数组

数组是具有一定顺序关系的若干相同类型变量的集合体，组成数组的变量称为该数组的元素。

数组属于构造类型。

# 1.1 一维数组的声明与引用

- 一维数组的声明

  类型说明符　数组名[ 常量表达式 ]；

  数组名的构成方法与一般变量名相同。

  例如：　int　a[10];

  表示 a 为整型数组，有10个元素：a[0]...a[9]

- 不能对数组大小做动态定义

- 引用

  必须先声明，后使用。

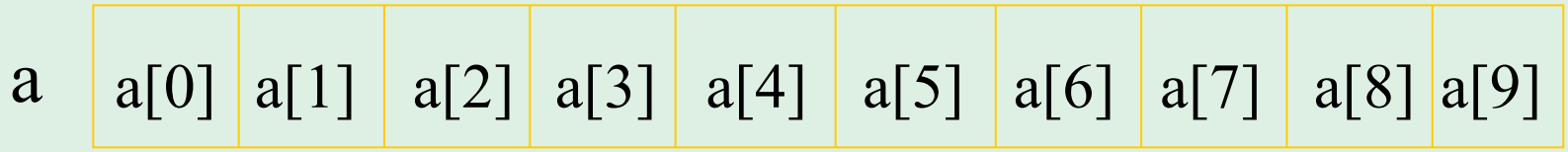
  只能逐个引用数组元素，而不能一次引用整个数组。

  例如：a[0]=a[5]+a[7]-a[2*3]

# 一维数组的存储顺序

数组元素在内存中顺次存放，它们的地址是连续的。

例如：具有10个元素的数组 a，在内存中的存放次序如下：




- 数组名字是数组首元素的内存地址。
- 数组名是一个常量，不能被赋值。

# 一维数组的初始化

可以在编译阶段使数组得到初值：

- 在声明数组时对数组元素赋以初值。
  例如：**static int a[10]={0,1,2,3,4,5,6,7,8,9};**
- 可以只给一部分元素赋初值。
  例如：**static int a[10]={0,1,2,3,4};**
- 在对全部数组元素赋初值时，可以不指定数组长度。
  例如：**static int a[]={1,2,3,4,5}**

# 一维数组应用举例

循环从键盘读入若干组选择题答案，计算并输出每组答案的正确率。

每组连续输入5个答案，每个答案可以是‘a’..‘d’。

```
#include <iostream>
using namespace std;
void main(void)
{   char  key[]= {'a','c','b','a','d'};
  char  c;
  int       ques = 0, numques = 5, numcorrect = 0;
  cout << "Enter the " << numques << " question tests:" << endl;
```

```
while (cin.get(c))
    {  if (c != '\n')
           if (c == key[ques])
           {    numcorrect++;
                cout << " ";
           }
           else      cout << "*";
       else
      {    cout<< " Score "<<float(numcorrect)/numques*100<<
    "%";
            ques = 0;            // reset variables
            numcorrect = 0;
            cout << endl;
            continue;
       }
       ques++;
    }
    system("pause");    return 0;
}
```

运行结果：

**acbba**

**** Score 60%**

**acbad**

**Score 100%**

**abbda**

*** ** Score 40%**

**bdcba**

******* Score 0%**

这个程序有没有问题？

# 1.2 二维数组的声明及引用

数据类型 标识符[常量表达式1][常量表达式2] …;

例:

```
int a[5][3];
```

表示a为整型二维数组，其中第一维有5个下标（0~4），第二维有3个下标（0~2），数组的元素个数为15，可以用于存放5行3列的整型数据表格。

# 二维数组的声明及引用

- 二维数组的声明

  类型说明符 数组名[常量表达式][常量表达式]

  例如：float a[3][4];

  可以理解为：

  a
  - a[0]——$a_{00}$ $a_{01}$ $a_{02}$ $a_{03}$
  - a[1]——$a_{10}$ $a_{11}$ $a_{12}$ $a_{13}$
  - a[2]——$a_{20}$ $a_{21}$ $a_{22}$ $a_{23}$

- 存储顺序

  按行存放，上例中数组a的存储顺序为：

  $a_{00}$ $a_{01}$ $a_{02}$ $a_{03}$ $a_{10}$ $a_{11}$ $a_{12}$ $a_{13}$ $a_{20}$ $a_{21}$ $a_{22}$ $a_{23}$

- 引用

  例如：b[1][2]=a[3][4]/2

# 二维数组的初始化

- 分行给二维数组赋初值

  例如：static int

  a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};

- 将所有数据写在一个{}内，按顺序赋值

  例如：static int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};

- 可以只对部分元素赋初值

  例如：int a[3][4]={{1},{0,6},{0,0,11}};

  int b[3[4]={{1}，{5}，{9}};

  它的作用是只对各行第1列的元素赋初值，其余元素值自动置为0。

例有一个3×4的矩阵，要求编程序求出其中值最大的那个元素的值，以及其所在的行号和列号。

```
#include <iostream>
using namespace std;
int main( )
{ int i,j,row=0,colum=0,max;
  int a[3][4]={{5,12,23,56},{19,28,37,46},{-12,-34,6,8}};
  max=a[0][0];                    //使max开始时取a［0］［0］的值
  for (i=0;i<=2;i++)              //从第0行~第2行
    for (j=0;j<=3;j++)            //从第0列~第3列
      if (a［i］［j］>max)           //如果某元素大于max
      {
       max=a［i］［j］;             //max将取该元素的值
       row=i;                   //记下该元素的行号i
       colum=j;                 //记下该元素的列号j
       }
cout<<"max="<<max<<",row="<<row<<",colum="<<colum<<endl
;
  return 0;
}
```

输出结果为

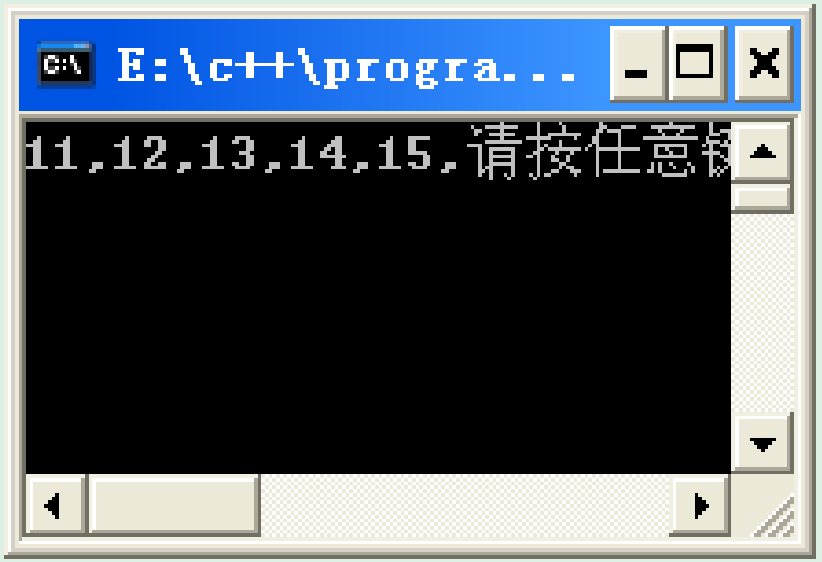
max=56，row=0，colum=3

# 1.3 数组作为函数参数

- 数组元素作实参，与单个变量一样。
- 数组名作参数，形、实参数都应是数组名，类型要一样，传送的是数组首地址。
- 数组名代表数组首元素的地址，并不代表数组中的全部元素。因此用数组名作函数实参时,不是把实参数组的值传递给形参，而只是将实参数组首元素的地址传递给形参。
- 对形参数组的改变会直接影响到实参数组。

# 例:对形参数组的改变会直接影响到实参数组。

```
#include "iostream"
using namespace std;
void f(int a[5])
{
    for ( int i=0;i<5;i++)
    a[i]=10+a[i];
    }
int main()
{
    int a[5]={1,2,3,4,5};
    f(a);
    for ( int i=0;i<5;i++)
    cout<<a[i]<<",";
    system("pause");
    return 0;
}
```



# 注意

- 形参一维数组的声明中可以写元素个数，也可以不写。
- C++实际上只把形参数组名作为一个指针变量来处理。
- 如果用二维数组名作为实参和形参，在对形参数组声明时，必须指定第二维(即列)的大小，且应与实参的第二维的大小相同。第一维的大小可以指定，也可以不指定。如

```
int array[3][10]          //形参数组的两个维都指定
或  int array[][10];       //第一维大小省略
```

- 但是不能把第二维的大小省略。下面的形参数组写法不合法：

  int array［3］［］；　　//不指定列数就无法确定数组的结构
- 在第二维大小相同的前提下，形参数组的第一维可以与实参数组不同。例如，实参数组定义为

  int score［5］［10］；

  而形参数组可以声明为

  int array［3］［10］；　//列数与实参数组相同，行数不同

  int array［8］［10］；
- 这时形参二维数组与实参二维数组都是由相同类型和大小的一维数组组成的，实参数组名score代表其首元素(即第一行)的起始地址，系统不检查第一维的大小。