第5章 数组

5.1 数组的概念

概括地说： 数组是有序数据的集合。要寻找一个数组中的某一个元素必须给出两个要素，即数组名和下标。数组名和下标惟一地标识一个数组中的一个元素。

数组是有类型属性的。同一数组中的每一个元素都必须属于同一数据类型。一个数组在内存中占一片连续的存储单元。如果有一个整型数组**a**，假设数组的起始地址为**2000**，则该数组在内存中的存储情况如图**5.1**所示。

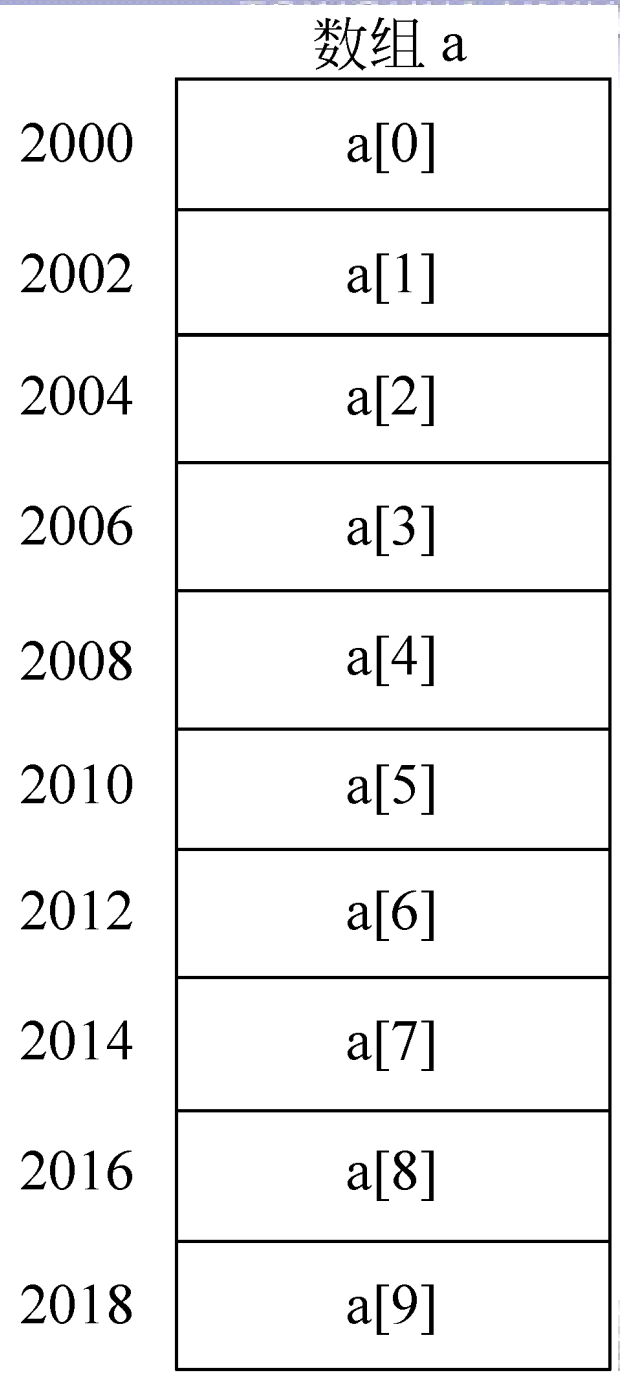
数组 a
2000
a[0]
2002
a[1]
2004
a[2]
2006
a[3]
2008
a[4]
2010
a[5]
2012
a[6]
2014
a[7]
2016
a[8]
2018
a[9]

图**5.1**

引入数组就不需要在程序中定义大量的变量，大大减少程序中变量的数量，使程序精炼，而且数组含义清楚，使用方便，明确地反映了数据间的联系。许多好的算法都与数组有关。熟练地利用数组，可以大大地提高编程和解题的效率，加强了程序的可读性。

C++用方括号来表示下标，如用**s［1］,s［2］,s［3］**分别代表**s1,s2,s3**。

5.2 一维数组的定义和引用

5.2.1 定义一维数组

定义一维数组的一般格式为

类型标识符 数组名［常量表达式］；

例如

int a［10］；

它表示数组名为**a**，此数组为整型，有**10**个元素。

说明：

(1) 数组名定名规则和变量名相同，遵循标识符定名规则。

(2) 用方括号括起来的常量表达式表示下标值，如下面的写法是合法的：

```
int a［10］;
int a［2*5］;
int a［n*2］;          //假设前面已定义了n为常变量
```

(3) 常量表达式的值表示元素的个数，即数组长度。例如，在“**int a［10］;**”中，**10**表示**a**数组有**10**个元素,下标从**0**开始,这**10**个元素是： **a［0］,a［1］,a［2］,a［3］,a［4］,a［5］,a［6］,a［7］, a［8］,a［9］**。注意最后一个元素是**a［9］**而不是**a［10］**。

(4) 常量表达式中可以包括常量、常变量和符号常量，但不能包含变量。也就是说，**C++**不允许对数组的大小作动态定义，即数组的大小不依赖于程序运行过程中变量的值。例如，下面这样定义数组是不行的：

```
int n;
cin>>n;                    //输入a数组的长度
int a［n］;                 //企图根据n的值决定数组的长度
```

如果把第**1**，**2**行改为下面一行就合法了：

```
const int n=5;
```

5.2.2 引用一维数组的元素

数组必须先定义，然后使用。只能逐个引用数组元素的值而不能一次引用整个数组中的全部元素的值。

数组元素的表示形式为

数组名［下标］

下标可以是整型常量或整型表达式。例如

a［0］= a［5］+ a［7］- a［2*3］

例**5.1** 数组元素的引用。

```
#include <iostream>
using namespace std;
int main( )
{int i,a［10］;
 for (i=0;i<=9;i++)
a［i］=i;
 for (i=9;i>=0;i--)
cout<<a［i］<<" ";
 cout<<endl;
 return 0;
}
```

运行结果如下：

```
9 8 7 6 5 4 3 2 1 0
```

程序使**a［0］~a［9］**的值为**0~9**，然后按逆序输出。

5.2.3 一维数组的初始化

(1) 在定义数组时分别对数组元素赋予初值。例如

int a［10］={0，1，2，3，4，5，6，7，8，9};

(2) 可以只给一部分元素赋值。例如

int a［10］={0,1,2,3,4};

(3) 如果想使一个数组中全部元素值为1，可以写成

int a［10］={1，1，1，1，1，1，1，1，1，1};

不能写成　　**int a［10］={1*10};**

不能给数组整体赋初值。

(4) 在对全部数组元素赋初值时，可以不指定数组长度。例如

int a［5］={1，2，3，4，5};

可以写成　　**int a［］={1，2，3，4，5};**

5.2.4 一维数组程序举例

例**5.2** 用数组来处理求**Fibonacci**数列问题。

可以用**20**个元素代表数列中的**20**个数，从第**3**个数开始，可以直接用表达式**f［i］=f［i-2］+f［i-1］**求出各数。

程序如下：

```
#include <iostream>
#include <iomanip>
using namespace std;
int main( )
  { int i;
    int f［20］={1,1};          //f［0］=1,f［1］=1
    for(i=2;i<20;i++)
```

```
        f［i］=f［i-2］+f［i-1］;    //在i的值为2时，f［2］=f［0］+f［1］，依此类推
    for(i=0;i<20;i++)              //此循环的作用是输出20个数
     {if(i%5==0) cout<<endl;       //控制换行，每行输出5个数据
       cout<<setw(8)<<f［i］;      //每个数据输出时占8列宽度
     }
    cout<<endl;                    //最后执行一次换行
    return 0;
}
```

运行结果如下：

（空一行）

```
   1       1       2       3       5
   8      13      21      34      55
  89     144     233     377     610
 987    1597    2584    4181    6765
```

例**5.3** 编写程序，用起泡法对**10**个数排序(按由小到大顺序)。

起泡法的思路是：将相邻两个数比较，将小的调到前头。见图**5.2**。

然后进行第**2**趟比较，对余下的前面**5**个数按上法进行比较，见图**5.3**。

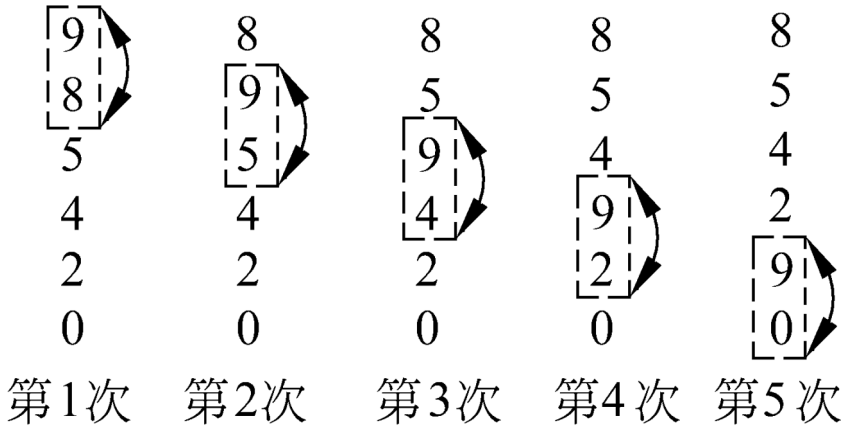

图**5.2**

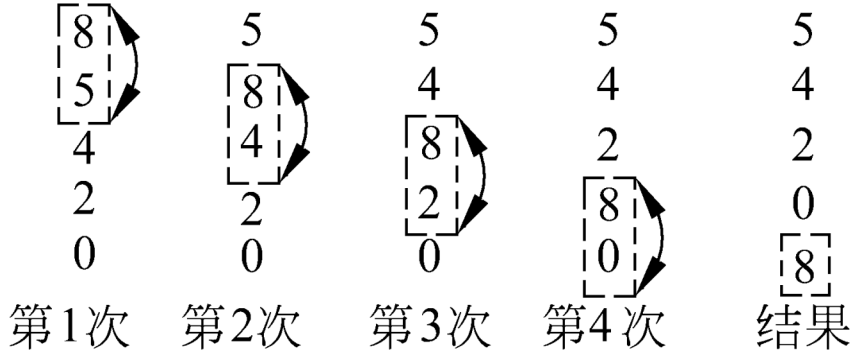

图**5.3**

可以推知，如果有**n**个数，则要进行**n-1**趟比较(和交换)。在第**1**趟中要进行**n-1**次两两比较，在第**j**趟中要进行**n-j**次两两比较。

根据以上思路写出程序，今设**n=10**，本例定义数组长度为**11**，**a**［**0**］不用，只用**a**［**1**］**~a**［**10**］，以符合人们的习惯。从前面的叙述可知，应该进行**9**趟比较和交换。

```
#include <iostream>
using namespace std;
int main( )
{
  int a［11］;
  int i,j,t;
  cout<<"input 10 numbers ： "<<endl;
  for (i=1;i<11;i++)                    //输入a［1］~a［10］
```

```
    cin>>a［i］;
  cout<<endl;
  for (j=1;j<=9;j++)                    //共进行9趟比较
    for(i=1;i<=10-j;i++)                //在每趟中要进行(10-j)次两两比较
      if (a［i］>a［i+1］)                  //如果前面的数大于后面的数
   {t=a［i］;a［i］=a［i+1］;a［i+1］=t;}  //交换两个数的位置，使小数上浮
  cout<<"the sorted numbers："<<endl;
  for(i=1;i<11;i++)                     //输出10个数
    cout<<a［i］<<" ";
  cout<<endl;
  return 0;
}
```

运行情况如下：

```
input 10 numbers:
3 5 9 11 33 6 -9 -76 100 123↙
the sorted numbers:
-76 -9 3 5 6 9 11 33 100 123
```

5.3 二维数组的定义和引用

具有两个下标的数组称为二维数组。有些数据要依赖于两个因素才能惟一地确定，例如有3个学生，每个学生有4门课的成绩，显然，成绩数据是一个二维表，如书中表5.1所示。

想表示第3个学生第4门课的成绩，就需要指出学生的序号和课程的序号两个因素，在数学上以$S_{3,4}$表示。在C++中以s［3］［4］表示，它代表数据73。

5.3.1 定义二维数组

定义二维数组的一般形式为

类型标识符 数组名［常量表达式］［常量表达式］

例如

float a［3］［4］,b［5］［10］;

定义**a**为**3×4(3**行**4**列**)**的单精度数组，**b**为**5×10(5**行**10**列**)**的单精度数组。注意不能写成**“float a［3,4］，b［5,10］;”**。**C++**对二维数组采用这样的定义方式，使我们可以把二维数组看作是一种特殊的一维数组：它的元素又是一个一维数组。例如，可以把**a**看作是一个一维数组，它有**3**个元素：**a［0］,a［1］,a［2］**，每个元素又是一个包含**4**个元素的一维数组，见图**5.4**。**a［0］,a［1］,a［2］**是**3**个一维数组的名字。

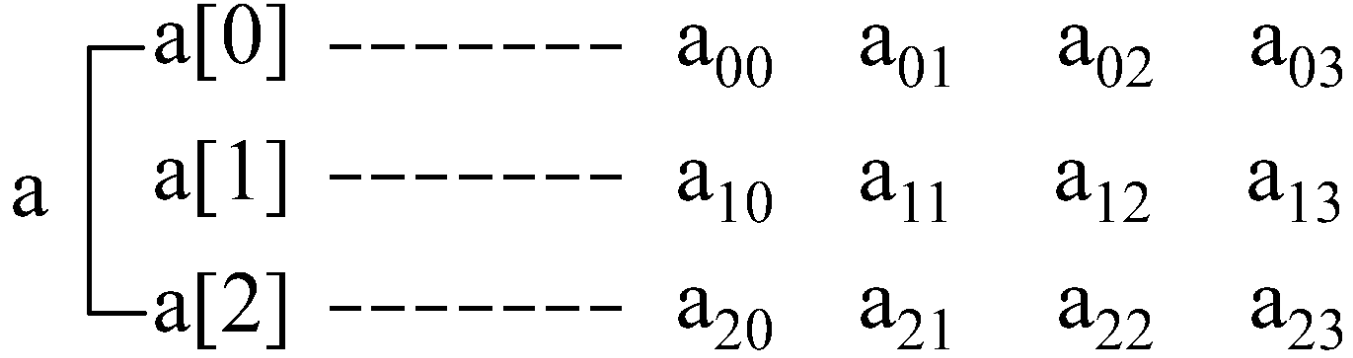

图**5.4**

上面定义的二维数组可以理解为定义了**3**个一维数组，即相当于

float a［0］［4］,a［1］［4］,a［2］［4］

此处把**a［0］,a［1］,a［2］**作一维数组名。**C++**的这种处理方法在数组初始化和用指针表示时显得很方便，这在以后会体会到。

C++中，二维数组中元素排列的顺序是：按行存放，即在内存中先顺序存放第一行的元素，再存放第二行的元素。图**5.5**表示对**a**［**3**］［**4**］数组存放的顺序。

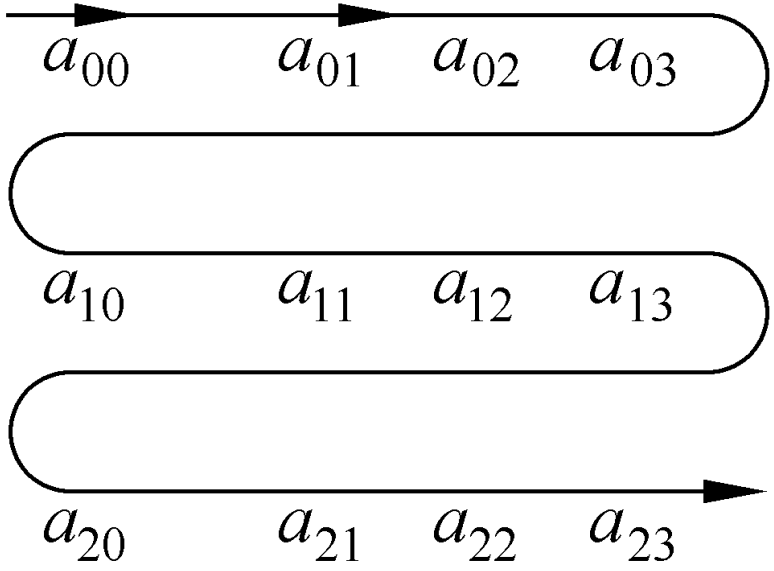

图**5.5**

C++允许使用多维数组。有了二维数组的基础，再掌握多维数组是不困难的。例如，定义三维数组的方法是

float a［2］［3］［4］;

定义**float**型三维数组**a**，它有**2×3×4=24**个元素。

多维数组元素在内存中的排列顺序：第一维的下标变化最慢，最右边的下标变化最快。例如，上述三维数组的元素排列顺序为

a［0］［0］［0］→a［0］［0］［1］→a［0］［0］［2］→a［0］［0］［3］→a［0］［1］［0］→a［0］［1］［1］→a［0］［1］［2］→a［0］［1］［3］→a［0］［2］［0］→a［0］［2］［1］→a［0］［2］［2］→a［0］［2］［3］→a［1］［0］［0］→a［1］［0］［1］→a［1］［0］［2］→a［1］［0］［3］→a［1］［1］［0］→a［1］［1］［1］→a［1］［1］［2］→a［1］［1］［3］→a［1］［2］［0］→a［1］［2］［1］→a［1］［2］［2］→a［1］［2］［3］

5.3.2 二维数组的引用

二维数组的元素的表示形式为

数组名［下标］［下标］

如**a［2］［3］**。下标可以是整型表达式，如**a［2-1］［2*2-1］**。不要写成**a［2,3］,a［2-1,2*2-1］**形式。

数组元素是左值，可以出现在表达式中，也可以被赋值，例如

```
b［1］［2］=a［2］［3］/2;
```

在使用数组元素时，应该注意下标值应在已定义的数组大小的范围内。常出现的错误是

```
int a［3］［4］;              //定义3行4列的数组
 ┆
a［3］［4］=15;               //引用a［3］［4］元素
```

定义a为3×4的数组，它可用的行下标值最大为2，列坐标值最大为3。最多可以用到a［2］［3］，a［3］［4］就超过了数组的范围。

请严格区分在定义数组时用的a［3］［4］和引用元素时的a［3］［4］的区别。前者a［3］［4］用来定义数组的维数和各维的大小，后者a［3］［4］中的3和4是下标值，a［3］［4］代表某一个元素。

5.3.3 二维数组的初始化

可以用下面的方法对二维数组初始化：

(1) 分行给二维数组赋初值。如

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

这种赋初值方法比较直观，把第**1**个花括号内的数据赋给第**1**行的元素，第**2**个花括号内的数据赋给第**2**行的元素……即按行赋初值。

(2) 可以将所有数据写在一个花括号内，按数组排列的顺序对各元素赋初值。如

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

效果与前相同。但以第**1**种方法为好，一行对一行，界限清楚。用第**2**种方法如果数据多，写成一大片，容易遗漏，也不易检查。

(3) 可以对部分元素赋初值。如

int a［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**。赋初值后数组各元素为

1 0 0 0
5 0 0 0
9 0 0 0

也可以对各行中的某一元素赋初值：

int a［3］［4］={{1},{0，6},{0,0,11}};

初始化后的数组元素如下：

1 0 0 0
0 6 0 0
0 0 11 0

这种方法对非0元素少时比较方便，不必将所有的0都写出来，只需输入少量数据。也可以只对某几行元素赋初值：

```
int a［3］［4］={{1}，{5，6}};
```

数组元素为

```
1 0 0 0
5 6 0 0
0 0 0 0
```

第3行不赋初值。也可以对第2行不赋初值：

```
int a［3］［4］={{1}，{}，{9}};
```

(4) 如果对全部元素都赋初值(即提供全部初始数据)，则定义数组时对第一维的长度可以不指定，但第二维的长度不能省。如

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

可以写成

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

系统会根据数据总个数分配存储空间，一共**12**个数据，每行**4**列，当然可确定为**3**行。

在定义时也可以只对部分元素赋初值而省略第一维的长度，但应分行赋初值。如

int a［］［4］={{0，0，3}，{}，{0，10}};

这样的写法，能通知编译系统： 数组共有**3**行。数组各元素为

0 0 3 0

0 0 0 0

0 10 0 0

C++在定义数组和表示数组元素时采用**a**［］［］这种两个方括号的方式，对数组初始化时十分有用，它使概念清楚，使用方便，不易出错。

5.3.4 二维数组程序举例

例**5.4** 将一个二维数组行和列元素互换，存到另一个二维数组中。例如

$$a=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \qquad b=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$$

程序如下：

```
#include <iostream>
using namespace std;
int main( )
{
 int a［2］［3］={{1,2,3},{4,5,6}};
 int b［3］［2］,i,j;
```

```
 cout<<"array a：  "<<endl;
 for (i=0;i<=1;i++)
 {
  for (j=0;j<=2;j++)
  { cout<<a［i］［j］<<" ";
    b［j］［i］=a［i］［j］;
  }
  cout<<endl;
 }
 cout<<"array b：  "<<endl;
 for (i=0;i<=2;i++)
 {
   for(j=0;j<=1;j++)
     cout<<b［i］［j］<<" ";
   cout<<endl;
  }
return 0;
}
```

运行结果如下：

```
array a:
1  2  3
4  5  6
array b:
1  4
2  5
3  6
```

例**5.5** 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。

开始时把**a**［**0**］［**0**］的值赋给变量**max**，然后让下一个元素与它比较，将二者中值大者保存在**max**中，然后再让下一个元素与新的**max**比，直到最后一个元素比完为止。**max**最后的值就是数组所有元素中的最大值。

程序如下：

```
#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

5.4 用数组名作函数参数

常量和变量可以用作函数实参，同样数组元素也可以作函数实参，其用法与变量相同。数组名也可以作实参和形参，传递的是数组的起始地址。

1. 用数组元素作函数实参

由于实参可以是表达式，而数组元素可以是表达式的组成部分，因此数组元素当然可以作为函数的实参，与用变量作实参一样，将数组元素的值传送给形参变量。

例**5.6** 用函数处理例**5.5**。

算法和例**5.5**是一样的，今设一函数**max_value**，用来进行比较并返回结果。可编写程序如下：

```
#include <iostream>
using namespace std;
int main( )
{ int max_value(int x,int max);              //函数声明
  int i,j,row=0,colum=0,max
  int a［3］［4］={{5,12,23,56},{19,28,37,46},{-12,-34,6,8}}; //数组初始化
    max=a［0］［0］;
  for (i=0;i<=2;i++)
   for (j=0;j<=3;j++)
{ max=max_value(a［i］［j］,max);        //调用max_value函数
if(max==a［i］［j］)                    //如果函数返回的是a［i］［j］的值
{row=i;                          //记下该元素行号i
```

```
            colum=j;                    //记下该元素列号j
        }
    }
    cout<<"max="<<max<<",row="<<row<<",colum="<<colum<<endl;
  }

int max_value(int x,int max)        //定义max_value函数
{if(x>max) return x;                //如果x>max，函数返回值为x
else return max;                    //如果x≤max，函数返回值为max
  }
```

2. 用数组名作函数参数

可以用数组名作函数参数，此时实参与形参都用数组名（也可以用指针变量，见第6章）。

例5.7 用选择法对数组中10个整数按由小到大排序。

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］到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轮。

根据此思路编写程序如下：

```
#include <iostream>
using namespace std;
int main( )
{void select_sort(int array［］,int n);          //函数声明
 int a［10］,i;
 cout<<"enter the originl array：  "<<endl;
```

```
for(i=0;i<10;i++)                          //输入10个数
  cin>>a［i］;
 cout<<endl;
 select_sort(a,10);                        //函数调用，数组名作实参
 cout<<"the sorted array：  "<<endl;
 for(i=0;i<10;i++)                         //输出10个已排好序的数
  cout<<a［i］<<"  ";
 cout<<endl;
 return 0;
}

void select_sort(int array［］,int n)          //形参array是数组名
{int i,j,k,t;
 for(i=0;i<n-1;i++)
{k=i;
    for(j=i+1;j<n;j++)
      if(array［j］<array［k］) k=j;
 t=array［k］;array［k］=array［i］;array［i］=t;
}
}
```

运行情况如下：

enter the originl array：

6 9 -2 56 87 11 -54 3 0 77↙ //输入10个数

the sorted array：

-54 -2 0 3 6 9 11 56 77 87

关于用数组名作函数参数有两点要说明：

(1) 如果函数实参是数组名，形参也应为数组名(或指针变量，关于指针见第**5**章)，形参不能声明为普通变量(如**int array;**)。实参数组与形参数组类型应一致（现都为**int**型），如不一致，结果将出错。

(2) 需要特别说明的是： 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。

形参可以是数组名，也可以是指针变量，它们用来接收实参传来的地址。如果形参是数组名，它代表的是形参数组首元素的地址。在调用函数时，将实参数组首元素的地址传递给形参数组名。这样，实参数组和形参数组就共占同一段内存单元。见图**5.6**。

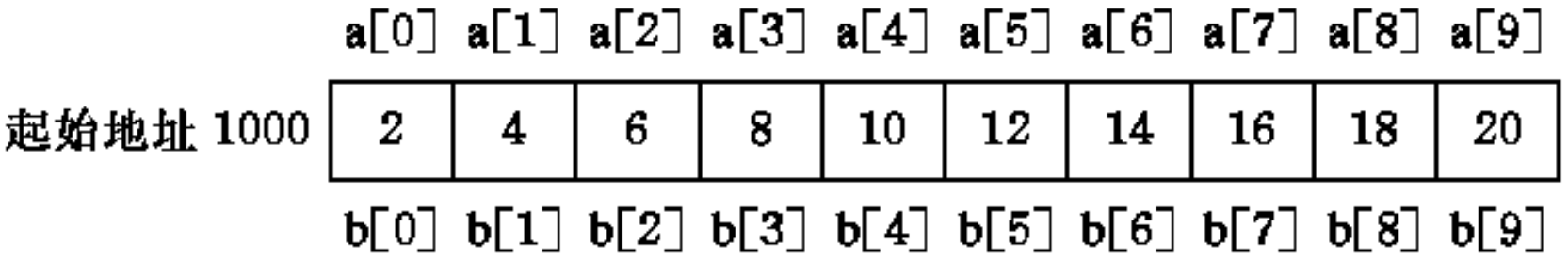

图**5.6**

在用变量作函数参数时，只能将实参变量的值传给形参变量，在调用函数过程中如果改变了形参的值，对实参没有影响，即实参的值不因形参的值改变而改变。而用数组名作函数实参时,改变形参数组元素的值将同时改变实参数组元素的值。在程序设计中往往有意识地利用这一特点改变实参数组元素的值。

实际上，声明形参数组并不意味着真正建立一个包含若干元素的数组，在调用函数时也不对它分配存储单元，只是用**array**［］这样的形式表示**array**是一维数组名，以接收实参传来的地址。因此**array**［］中方括号内的数值并无实际作用，编译系统对一维数组方括号内的内容不予处理。形参一维数组的声明中可以写元素个数，也可以不写。

函数首部的下面几种写法都合法，作用相同。

```
void select_sort(int array［10］,int n)    //指定元素个数与实参数组相同
void select_sort(int array［］,int n)        //不指定元素个数
void select_sort(int array［5］,int n)     //指定元素个数与实参数组不同
```

在学习第6章时可以进一步知道，C++实际上只把形参数组名作为一个指针变量来处理，用来接收从实参传过来的地址。前面提到的一些现象都是由此而产生的。

3. 用多维数组名作函数参数

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
```

二者都合法而且等价。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［］［］;          //不能确定数组的每一行有多少列元素
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。

如果是三维或更多维的数组，处理方法是类似的。

例**5.8** 有一个３×４的矩阵，求矩阵中所有元素中的最大值。要求用函数处理。

解此题的算法已在例**5.5**中介绍。

程序如下：

```
#include <iostream>
using namespace std;
int main( )
{int max_value(int array［］［4］);
 int a［3］［4］={{11,32,45,67},{22,44,66,88},{15,72,43,37}};
 cout<<"max value is "<<max_value(a)<<endl;
 return 0;
}

int max_value(int array［］［4］)
{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］;
 return max;
}
```

运行结果如下：

```
max value is 88
```

读者可以将**max_value**函数的首部改为以下几种情况，观察编译情况：

```
int max_value(int array［］［］)
int max_value(int array［3］［］)
int max_value(int array［3］［4］)
int max_value(int array［10］［10］)
int max_value(int array［12］)
```

5.5 字符数组

用来存放字符数据的数组是字符数组，字符数组中的一个元素存放一个字符。字符数组具有数组的共同属性。由于字符串应用广泛，C和C++专门为它提供了许多方便的用法和函数。

5.5.1 字符数组的定义和初始化

定义字符数组的方法与前面介绍的类似。例如

```
char c［10］;
c［0］='I';c［1］=' ';c［2］='a';c［3］='m';c［4］=' ';c［5］='h';c［6］='a';c［7］='p';c［8］='p';
 c［9］='y';
```

上面定义了**c**为字符数组，包含**10**个元素。在赋值以后数组的状态如图**5.7**所示。

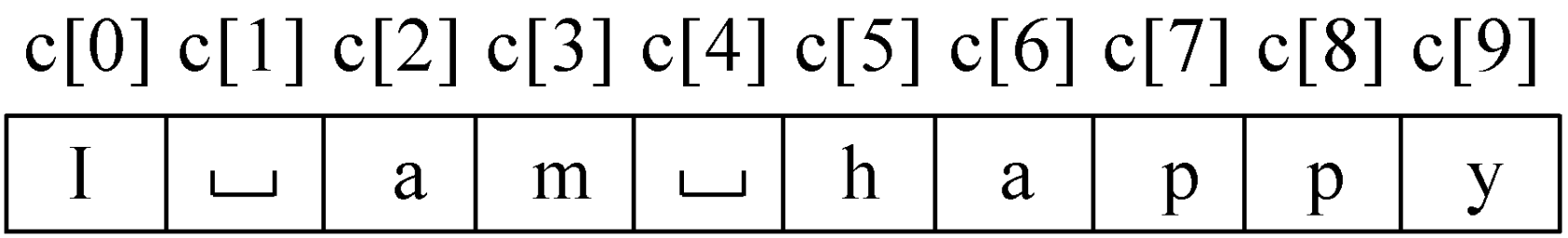

图**5.7**

对字符数组进行初始化，最容易理解的方式是逐个字符赋给数组中各元素。如

```
char c［10］={'I',' ','a','m',' ','h','a','p','p','y'};
```

把10个字符分别赋给c［0］~c［9］这10个元素。

如果花括号中提供的初值个数大于数组长度，则按语法错误处理。如果初值个数小于数组长度，则只将这些字符赋给数组中前面那些元素，其余的元素自动定为空字符。如果提供的初值个数与预定的数组长度相同，在定义时可以省略数组长度，系统会自动根据初值个数确定数组长度。如

```
char c［］={'I',' ','a','m',' ','h','a','p','p','y'};
```

也可以定义和初始化一个二维字符数组，如

```
char diamond［5］［5］={{' ',' ','*'},{' ','*', ' ', '*'}, {'*', ' ', ' ', ' ', '*'}, {' ', '*', ' ', '*'}, {' ', ' ', '*'}};
```

5.5.2 字符数组的赋值与引用

只能对字符数组的元素赋值，而不能用赋值语句对整个数组赋值。如

```
char c［5］;
c={'C','h','i','n','a'};          //错误，不能对整个数组一次赋值
c［0］='C'; c［1］='h';c［2］='i';c［3］='n';c［4］='a'; //对数组元素赋值，正确
```

如果已定义了a和b是具有相同类型和长度的数组，且b数组已被初始化，请分析：

```
a=b;                              //错误，不能对整个数组整体赋值
a［0］=b［0］;                     //正确，引用数组元素
```

例**5.9** 设计和输出一个钻石图形。

```
#include <iostream>
using namespace std;
void main( )
{char diamond［］［5］={{' ',' ','*'},{' ','*',' ','*'},{'*',' ',' ',' ','*'},
{' ','*',' ',' ','*'},{' ',' ','*'}};
    int i,j;
    for (i=0;i<5;i++)
      {for (j=0;j<5;j++)
cout<<diamond［i］［j］;        //逐个引用数组元素，每次输出一个字符
cout<<endl;
}
}
```

运行结果为

```
   *
  * *
*     *
  * *
   *
```

5.5.3 字符串和字符串结束标志

用一个字符数组可以存放一个字符串中的字符。如

```
char str［12］={'I',' ','a','m',' ','h','a','p','p','y'};
```

用一维字符数组**str**来存放一个字符串**"I am happy"**中的字符。字符串的实际长度**(10)**与数组长度**(12)**不相等，在存放上面**10**个字符之外，系统对字符数组最后两元素自动填补空字符**'\0'**。

为了测定字符串的实际长度，**C++**规定了一个“字符串结束标志”，以字符**'\0'**代表。在上面的数组中，第**11**个字符为**'\0'**，就表明字符串的有效字符为其前面的**10**个字符。也就是说，遇到字符**'\0'**就表示字符串到此结束，由它前面的字符组成字符串。

对一个字符串常量，系统会自动在所有字符的后面加一个'\0'作为结束符。例如字符串"I am happy"共有10个字符，但在内存中它共占11个字节，最后一个字节'\0'是由系统自动加上的。

在程序中往往依靠检测'\0'的位置来判定字符串是否结束，而不是根据数组的长度来决定字符串长度。当然，在定义字符数组时应估计实际字符串长度，保证数组长度始终大于字符串实际长度。如果在一个字符数组中先后存放多个不同长度的字符串，则应使数组长度大于最长的字符串的长度。

说明：'\0'只是一个供辨别的标志。

如果用以下语句输出一个字符串：

```
cout<<"
How do you do?";
```

系统在执行此语句时逐个地输出字符，那么它怎么判断应该输出到哪个字符就停止了呢?

下面再对字符数组初始化补充一种方法：用字符串常量来初始化字符数组。例如

```
char str［］={"I  am  happy"};
```

也可以省略花括号，直接写成

```
char str［］="I am happy";
```

不是用单个字符作为初值，而是用一个字符串(注意字符串的两端是用双撇号而不是单撇号括起来的)作为初值。显然，这种方法直观，方便，符合人们的习惯。注意：数组str的长度不是10，而是11(因为字符串常量的最后由系统加上一个'\0')。因此，上面的初始化与下面的初始化等价：

```
char str［］={'I',' ','a','m',' ','h','a','p','p','y','\0'};
```

而不与下面的等价：

char str［］={'I',' ','a','m',' ','h','a','p','p','y'};

前者的长度为**11**，后者的长度为**10**。如果有

char str［10］="China";

数组**str**的前**5**个元素为**'C','h','i','n','a'**，第**6**个元素为**'\0'**，后**4**个元素为空字符。见图**5.8**。

C	h	i	n	a	\0	\0	\0	\0	\0

图**5.8**

需要说明的是：字符数组并不要求它的最后一个字符为'\0'，甚至可以不包含'\0'。如以下这样写完全是合法的：

```
char str［5］={'C','h','i','n','a'};
```

是否需要加'\0'，完全根据需要决定。但是由于C++编译系统对字符串常量自动加一个'\0'。因此，人们为了使处理方法一致，便于测定字符串的实际长度，以及在程序中作相应的处理，在字符数组中有效字符的后面也人为地加上一个'\0'。如

```
char str［6］={'C','h','i','n','a','\0'};
```

5.5.4 字符数组的输入输出

字符数组的输入输出可以有两种方法：

(1) 逐个字符输入输出，如例**5.9**。

(2) 将整个字符串一次输入或输出。例如有以下程序段：

```
char str［20］;
cin>>str;                    //用字符数组名输入字符串
cout<<str;                   //用字符数组名输出字符串
```

在运行时输入一个字符串，如

China↙

在内存中，数组**str**的状态如图**5.9**所示，在**5**个字符的后面自动加了一个结束符**′\0′**。

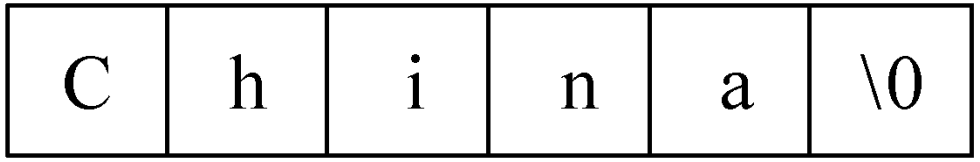

图5.9

输出时，逐个输出字符直到遇结束符'\0'，就停止输出。输出结果为

China

如前所述，字符数组名**str**代表字符数组第一个元素的地址，执行**“cout<<str;”**的过程是从**str**所指向的数组第一个元素开始逐个输出字符，直到遇到'\0'为止。

请注意：

(1) 输出的字符不包括结束符'\0'。

(2) 输出字符串时，**cout**流中用字符数组名，而不是数组元素名。

(3) 如果数组长度大于字符串实际长度，也只输出到遇**′\0′**结束。

(4) 如果一个字符数组中包含一个以上**′\0′**，则遇第一个**′\0′**时输出就结束。

(5) 用**cin**从键盘向计算机输入一个字符串时，从键盘输入的字符串应短于已定义的字符数组的长度，否则会出现问题。

C++提供了**cin**流中的**getline**函数，用于读入一行字符**(**或一行字符中前若干个字符**)**，使用安全又方便，请参阅第**13**章**13.3.2**节。

5.5.5 字符串处理函数

由于字符串使用广泛，C和C++提供了一些字符串函数，使得用户能很方便地对字符串进行处理。几乎所有版本的C++都提供下面这些函数，它们是放在函数库中的，在string和string.h头文件中定义。如果程序中使用这些字符串函数，应该用#include命令把string.h或string头文件包含到本文件中。下面介绍几种常用的函数。

1. 字符串连接函数 strcat

其函数原型为

strcat(char［］,const char［］);

strcat是**string catenate**（字符串连接）的缩写。该函数有两个字符数组的参数，函数的作用是：将第二个字符数组中的字符串连接到前面字符数组的字符串的后面。第二个字符数组被指定为**const**，以保证该数组中的内容不会在函数调用期间修改。连接后的字符串放在第一个字符数组中，函数调用后得到的函数值，就是第一个字符数组的地址。例如

```
char str1［30］="People's Republic of ";
char str2［］="China";
cout<<strcat(str1，str2));                    //调用strcat函数
```

输出：

People′s Republic of China

连接前后的状况如图**5.10**所示。

tr1:	P	e	o	p	l	e	'	s	␣	R	e	p	u	b	l	i	c	␣	o	f	␣	\0	\0	\0	\0	\0	\0
tr2:	C	h	i	n	a	\0																					
tr3:	P	e	o	p	l	e	'	s	␣	R	e	p	u	b	l	i	c	␣	o	f	␣	C	h	i	n	a	\0

图5.10

2. 字符串复制函数strcpy

其函数原型为

strcpy(char［］,const char［］);

strcpy是**string copy**（字符串复制）的缩写。它的作用是将第二个字符数组中的字符串复制到第一个字符数组中去，将第一个字符数组中的相应字符覆盖。例如

char str1［10］，str2［］="China";

strcpy(str1，str2);

执行后，**str2**中的**5**个字符**"China"**和**'＼0'**(共**6**个字符)复制到数组**str1**中。

说明：

(1) 在调用**strcpy**函数时，第一个参数必须是数组名(如**str1**)，第二个参数可以是字符数组名，也可以是一个字符串常量。

(2) 可以用**strcpy**函数将一个字符串中前若干个字符复制到字符数组中去。

(3) 只能通过调用**strcpy**函数来实现将一个字符串赋给一个字符数组，而不能用赋值语句将一个字符串常量或字符数组直接赋给一个字符数组。

3. 字符串比较函数strcmp

其函数原型为

strcmp(const char［］,const char［］);

strcmp是**string compare**（字符串比较）的缩写。作用是比较两个字符串。由于这两个字符数组只参加比较而不应改变其内容，因此两个参数都加上**const**声明。以下写法是合法的：

strcmp(str1，str2);

strcmp("China"，"Korea");

strcmp(str1，"Beijing");

比较的结果由函数值带回。

(1) 如果字符串**1=**字符串**2**，函数值为**0**。

(2) 如果字符串**1>**字符串**2**，函数值为一正整数。

(3) 如果字符串**1<**字符串**2**，函数值为一负整数。

字符串比较的规则与其他语言中的规则相同，即对两个字符串自左至右逐个字符相比(按**ASCII**码值大小比较)，直到出现不同的字符或遇到**'\0'**为止。如全部字符相同，则认为相等；若出现不相同的字符，则以第一个不相同的字符的比较结果为准。

注意：对两个字符串比较，不能用以下形式：

```
if(str1>str2) cout<<"yes";
```

字符数组名**str1**和**str2**代表数组地址，上面写法表示将两个数组地址进行比较，而不是对数组中的字符串进行比较。对两个字符串比较应该用

```
if(strcmp(str1，str2)>0) cout<<"yes";
```

4. 字符串长度函数strlen

函数原型为

strlen(const char［］);

strlen是**string length**（字符串长度）的缩写。它是测试字符串长度的函数。其函数的值为字符串中的实际长度，不包括**'\0'**在内。如

```
char str［10］="China";
cout<<strlen(str);
```

输出结果不是**10**，也不是**6**，而是**5**。

以上是几种常用的字符串处理函数，除此之外还有其他一些函数。

5.5.6 字符数组应用举例

例5.10 有3个字符串,要求找出其中最大者。要求用函数调用。

程序如下：

```
#include <iostream>
#include <string>
using namespace std;
int main( )
{ void max_string(char str［］［30］,int i);      //函数声明
int i;
  char country_name［3］［30］;
  for(i=0;i<3;i++)
     cin>>country_name［i］;                    //输入3个国家名
  max_string(country_name,3);                  //调用max_string函数
  return 0;
```

```
}

void max_string(char str［］［30］,int n)
{
  int i;
  char string［30］;
  strcpy(string,str［0］);                    //使string的值为str［0］的值
  for(i=0;i<n;i++)
     if(strcmp(str［i］,string)>0)              //如果str［i］>string
strcpy(string,str［i］);              //将str［i］中的字符串复制到string
  cout<<endl<<"the largest string is：  "<<string<<endl;
}
```

运行结果如下：

```
CHINA↙
GERMANY↙
FRANCH↙
the largest string is：  GERMANY
```

*5.6 C++处理字符串的方法——字符串类与字符串变量

用字符数组来存放字符串并不是最理想和最安全的方法。

C++提供了一种新的数据类型——字符串类型(**string**类型)，在使用方法上，它和**char**、**int**类型一样，可以用来定义变量，这就是字符串变量——用一个名字代表一个字符序列。

实际上，**string**并不是**C++**语言本身具有的基本类型，它是在**C++**标准库中声明的一个字符串类，用这种类可以定义对象。每一个字符串变量都是**string**类的一个对象。

5.6.1 字符串变量的定义和引用

1. 定义字符串变量

和其他类型变量一样，字符串变量必须先定义后使用，定义字符串变量要用类名**string**。如

```
string string1;                    //定义string1为字符串变量
string string2="China";            //定义string2同时对其初始化
```

应当注意： 要使用**string**类的功能时，必须在本文件的开头将**C++**标准库中的**string**头文件包含进来，即应加上

```
#include <string>                  //注意头文件名不是string.h
```

2. 对字符串变量的赋值

在定义了字符串变量后，可以用赋值语句对它赋予一个字符串常量，如

```
string1="Canada";
```

既可以用字符串常量给字符串变量赋值，也可以用一个字符串变量给另一个字符串变量赋值。如

```
string2=string1;          //假设string2和string1均已定义为字符串变量
```

不要求**string2**和**string1**长度相同，假如**string2**原来是**"China"**，**string1**原来是**"Canada"**，赋值后**string2**也变成**"Canada"**。在定义字符串变量时不需指定长度，长度随其中的字符串长度而改变。

可以对字符串变量中某一字符进行操作，如

```
string word="Then";       //定义并初始化字符串变量word
word［2］='a'; //修改序号为2的字符，修改后word的值为"Than"
```

3. 字符串变量的输入输出

可以在输入输出语句中用字符串变量名，输入输出字符串，如

```
cin>> string1;          //从键盘输入一个字符串给字符串变量string1
cout<< string2;         //将字符串string2输出
```

5.6.2 字符串变量的运算

在上一节中可以看到： 在以字符数组存放字符串时，字符串的运算要用字符串函数，如**strcat**(连接)、**strcmp**(比较)、**strcpy**(复制)，而对**string**类对象，可以不用这些函数，而直接用简单的运算符。

(1) 字符串复制用赋值号

```
string1=string2;
```

其作用与“**strcpy(string1,string2);**”相同。

(2) 字符串连接用加号

```
string string1="C++";                  //定义string1并赋初值
string string2="Language";             //定义string2并赋初值
string1=string1 + string2;             //连接string1和string2
```

连接后**string1**为**"C++ Language"**。

(3) 字符串比较直接用关系运算符

可以直接用 **==(**等于**)**、**>(**大于**)**、**<(**小于**)**、**!=(**不等于**)**、**>=(**大于或等于**)**、**<=(**小于或等于**)**等关系运算符来进行字符串的比较。

使用这些运算符比使用**5.5.5**节中介绍的字符串函数直观而方便。

5.6.3 字符串数组

不仅可以用**string**定义字符串变量，也可以用**string**定义字符串数组。如

```
string name［5］;          //定义一个字符串数组，它包含5个字符串元素
string name［5］={"Zhang","Li","Fun","Wang","Tan"};
//定义一个字符串数组并初始化
```

此时**name**数组的状况如图**5.11**所示。

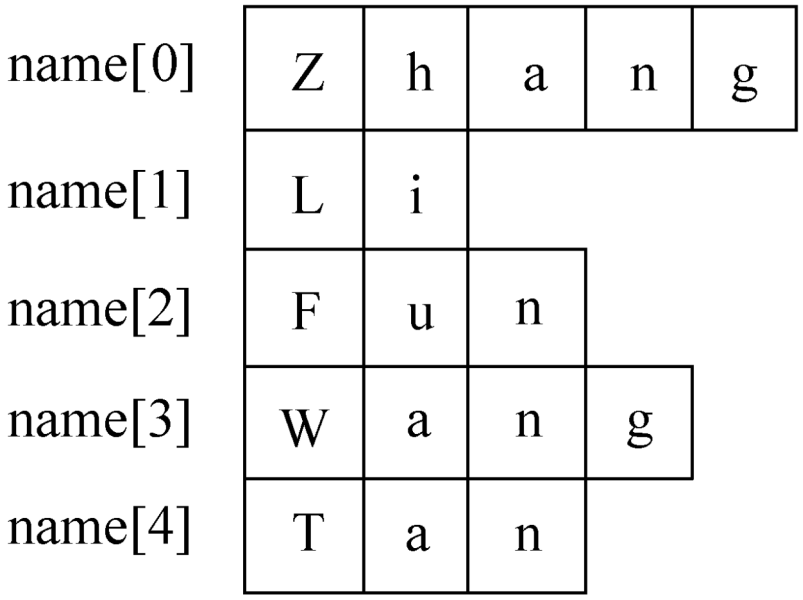

图**5.11**

可以看到：

(1) 在一个字符串数组中包含若干个**(**现为**5**个**)**元素，每个元素相当于一个字符串变量。

(2) 并不要求每个字符串元素具有相同的长度，即使对同一个元素而言，它的长度也是可以变化的，当向某一个元素重新赋值，其长度就可能发生变化。

(3) 在字符串数组的每一个元素中存放一个字符串，而不是一个字符，这是字符串数组与字符数组的区别。如果用字符数组存放字符串，一个元素只能存放一个字符，用一个一维字符数组存放一个字符串。

(4) 每一个字符串元素中只包含字符串本身的字符而不包括**′\0′**。

可见用字符串数组存放字符串以及对字符串进行处理是很方便的。

在定义字符串数组时怎样给数组分配存储空间呢？实际上，编译系统为每一个字符串变量分配**4**个字节，在这个存储单元中，并不是直接存放字符串本身，而是存放字符串的地址。在本例中，就是把字符串**"Zhang"**的地址存放在**name**［**0**］，把字符串**"Li"** 的地址存放在**name**［**1**］，把字符串**"Fun"**的地址存放在**name**［**2**］**……**图**5.11**只是一个示意图。在字符串变量中存放的是字符串的指针**(**字符串的地址**)**。

5.6.4 字符串运算举例

例5.11 输入3个字符串，要求将字母按由小到大的顺序输出。

```
#include <iostream>
#include <string>
using namespace std;
int main( )
{string string1,string2,string3,temp;
 cout<<"please input three strings:  ";  //这是对用户输入的提示
 cin>>string1>>string2>>string3;       //输入3个字符串
 if(string2>string3) {temp=string2;string2=string3;string3=temp;}
//使串2≤串3
 if(string1<=string2) cout<<string1<<" "<<string2<<" "<<string3<<endl;
   //如果串1≤串2，则串1≤串2≤串3
```

```
 else if(string1<=string3) cout<<string2<<" "<<string1<<"
"<<string3<<endl;
   //如果串1＞串2，且串1≤串3，则串2＜串1≤串3
 else cout<<string2<<" "<<string3<<" "<<string1<<endl;
   //如果串1＞串2，且串1＞串3，则串2＜串3＜串1
}
```

运行情况如下：

```
please input three strings：  China  U.S.A. Germany↙
China Germany U.S.A.
```

例**5.12** 一个班有**n**个学生，需要把每个学生的简单材料(姓名和学号)输入计算机保存。然后可以通过输入某一学生的姓名查找其有关资料。当输入一个姓名后，程序就查找该班中有无此学生，如果有，则输出他的姓名和学号，如果查不到，则输出“本班无此人”。

为解此问题，可以分别编写两个函数，函数**input_data**用来输入**n**个学生的姓名和学号，函数**search**用来查找要找的学生是否在本班。

程序可编写如下：

```
#include <iostream>
#include <string>
using namespace std;
string name［50］,num［50］;          //定义两个字符串数组，分别存放姓名和学号
int n;                          //n 是实际的学生数
```

```
int main( )
{void input_data( );                  // 函数声明
 void search(string find_name);       //函数声明
 string find_name;                    //定义字符串变量，find_name是要找的学生
 cout<<"please input number of this class： "; //输入提示： 请输入本班学生的人数
 cin>>n;                              // 输入学生数
 input_data( );                       // 调用input_data函数，输入学生数据
 cout<<"please input name you want find： "; //输入提示： 请输入你要找的学生名
 cin>>find_name;                      // 输入要找的学生的姓名
 search(find_name);                   // 调用search函数，寻找该学生姓名
 return 0;
}

void input_data( )                    // 函数首部
{int i;
 for (i=0;i<n;i++)
  {cout<<"input name and NO. of student "<<i+1<<"： ";
//输入提示
   cin>>name［i］>>num［i］;}          // 输入n个学生的姓名和学号
}
```

```
void search(string find_name)              // 函数首部
{int i;
 bool flag=false;
 for(i=0;i<n;i++)
  if(name［i］==find_name)                 //如果要找的姓名与本班某一学生姓名相同
   { cout<<name［i］<<" has been found, his number is " <<num［i］<<endl;
//输出姓名与学号
flag=true;
break;
}
 if(flag==false) cout<<"can't find this name";//如找不到，输出“找不到”的信息
}
```

运行情况如下：

```
please input number of this class：5↙
input name and number of student 1：Li 1001↙
input name and number of student 2：Zhang 1002↙
input name and number of student 3：Wang 1003↙
input name and number of student 4：Tan 1004↙
input name and number of student 5：Fun 1005↙
please input name you want find：Wang↙
Wang has been found，his number is 1003
```

请考虑：

(1) 程序第**3**行定义全局变量时，数组的大小不指定为**50**，而用变量**n**，即**string name［n］,num［n］;n**在运行时输入，行不行？为什么？

(2) search函数**for**循环中最后有一个**break**语句，它起什么作用？不要行不行？

(3) 如果不使用全局变量，把变量**n**和数组**name**，**num**都作为局部变量，通过虚实结合的方法在函数间传递数据，这样行不行？请思考并上机试一下。

通过以上两个例子可以看到，用**string**定义字符串变量，简化了操作，把原来复杂的问题简单化了，这是**C++**对**C**的一个发展。

归纳起来，**C++**对字符串的处理有两种方法： 一种是用字符数组的方法，这是**C**语言采取的方法， 一般称为**C string**方法；一种是用**string**类定义字符串变量，称为**string**方法。显然，**string**方法概念清楚，使用方便，最好采用这种方法。**C++**保留**C-string**方法主要是为了与**C**兼容，使以前用**C**写的程序能用于**C++**环境。