# 第5章 数　组

**本章要点**

- 数组的概念。
- 一维数组、二维数组的声明及初始化。
- 数组作为函数的参数。
- 字符串与数组的关系。
- 数组的应用(排序、查找、统计、字符处理和数列处理等)。

在程序中,经常需要保存大量具有相同类型的数据,如在程序中保存 10 个学生的考试成绩,并对成绩进行排序、统计等操作。最直接的想法是定义 10 个变量 n1,n2,n3,…,n10 用于存放考试成绩,且不说如何对 10 个变量进行排序,就是为这 10 个变量赋初值就比较麻烦。这只是 10 个数据,如果是 1000 个或更多的数据,对这些数据操作的复杂程度可想而知。为解决上述问题,C++ 提供了一个很好的复合类型——数组。数组与循环控制结构配合使用,可以很方便地解决对大量数据赋值、排序、统计等操作问题。

## 5.1 数组的基本概念

数组是具有相同类型的一组数据的集合,且占用连续的内存单元用于存储信息。组成数组的对象称为数组元素,可以通过指定数组元素在数组中的位置编号来访问数组中的元素。

例如,定义数组 score[]用于存放 10 个学生的成绩:

```
int score[10] = {78,66,93,72,84,90,67,81,86,95};
```

表 5-1 列出了整型数组 score[]的各数组元素及其值。

**表 5-1　数组 score[]的各数组元素及其值**

| 数 组 元 素 | 数组元素的值 | 数 组 元 素 | 数组元素的值 |
|---|---|---|---|
| score [0] | 78 | score [5] | 90 |
| score [1] | 66 | score [6] | 67 |
| score [2] | 93 | score [7] | 81 |
| score [3] | 72 | score [8] | 86 |
| score [4] | 84 | score [9] | 95 |

score 数组包含 10 个元素,可以用数组名加上方括号“[]”中该元素的位置编号访问这

些元素。例如,score[]数组中的第1个元素为score[0],第2个元素为score[1],……,第10个元素为score[9]。由此可见,score[]数组中的第i个元素为score[i-1]。

在实际应用中,数组分为简单的一维数组和较为复杂的多维数组。

# 5.2 一维数组

## 5.2.1 一维数组的声明

数组在使用前必须先声明。声明一维数组的形式如下:

**<类型标识符><数组名>[数组长度]**

其中,类型标识符决定数组中每一个数组元素可存储数据的类型,一个数组中,所有数组元素的类型都是相同的;数组名必须遵循C++语言对标识符的要求,其命名规则与其他变量名相同;数组长度是个常量表达式,它规定了数组的大小,即所声明的数组由多少个数据类型相同的存储空间组成。

数组的声明为数组分配了存储空间,数组中每个元素在内存中是依次排列的。例如:

```
int score[10];
```

定义了名称为score的一维数组,该数组有10个元素,且每一个数组元素都是int类型的,即每一个数组元素占4字节,所以该数组在内存中一共占4×10=40字节,且这些存储空间是连续的。

在声明数组时,要注意数组的长度只能由常量表达式来决定,不能有变量出现,即数组的长度必须是确定的。如果有变量出现,在编译时编译器会给出错误信息。例如:

```
int const N = 10;
int n,m = 3;
int a[N];                          //合法,N为常变量
double b[n],c[m * 12];             //不合法,n和m为变量
```

## 5.2.2 一维数组的初始化

变量可以在声明时赋初值,数组也可以在声明时给所有或部分数组元素赋初始值。声明时给一维数组元素赋初始值有如下两种形式。

形式1:

**<类型标识符> <数组名>[数组长度]={第0个元素值,第1个元素值,…,第n-1个元素值}**

形式2:

**<类型标识符> <数组名>[]={第0个元素值,第1个元素值,…,第n-1个元素值}**

第一种形式将声明一个长度为“数组长度”的数组,然后将花括号内的值依次赋予数组的各个元素。花括号内只能是常量表达式。如果花括号中的常量表达式的个数少于数组长度,则剩余的数组元素就不被赋予初始值,都被“清零”;如果花括号中的常量表达式的个数大于数组长度,则编译器会给出错误信息。

第二种形式将声明一个长度为 n 的数组，并将花括号内的 n 个值依次赋给数组的各个元素。注意，此时数组的长度未给定，而是由花括号内的数据个数决定的。

特别提示：以下形式是不允许的。

**<类型标识符><数组名>[ ] = {第 0 个元素值,,第 2 个元素值,…,第 n－1 个元素值}**

例如：

```
int a[ ] = {1,2,3};              //合法,默认数组长度为 3,a[0] = 1,a[1] = 2,a[2] = 3
int b[5] = {1,2,3};              //合法,b[0] = 1,b[1] = 2,b[2] = 3,b[3] = b[4] = 0
int c[ ] = {1,,3,4,5};           //不合法,1 和 3 之间不能有"空"
int d[6] = {1,2,3,4,5,6,7};      //不合法,初值个数 7 >数组长度 6
```

### 5.2.3 访问一维数组的元素

带下标的数组名就是下标变量，也称为数组元素。对数组的操作一般都是通过访问数组元素来实现的。访问一维数组元素的形式如下：

**<数组名>[下标]**

下标就是数组元素的索引值，即数组元素在数组中的位置编号，它代表了要访问的数组元素在数组中与第 1 个数组元素(下标值为 0)的相对位置。例如，数组 a[]中第 1 个数组元素 a[0]的下标为 0，因为就是它本身，所以它与第 1 个数组元素 a[0]的相对位置为 0；第 2 个数组元素 a[1]的下标为 1，代表它与第 1 个数组元素 a[0]的相对位置为 1，即 a[1]是 a[0]的下一个元素；第 3 个数组元素 a[2]的下标为 2，表示它与第 1 个数组元素 a[0]的相对位置为 2，以此类推。

下标值的允许范围是 0～N－1(N 为数组的长度)。在声明数组时，数组的长度只能是常量，而在访问数组元素时，下标可以是任意的整型表达式，只要表达式的取值范围在 0～N－1 即可。

在实际的程序设计中，要特别注意下标值的溢出问题。C++在编译时不对下标值的合法性进行检查。换句话说，如果声明了一个长度为 100 的数组，访问下标为 0、下标为－1、下标为 120 的元素，在语法上都是正确的，这就要求程序员在编写程序时要格外小心。如果访问数组中的元素时下标值在允许的范围以外，就等于企图侵入程序中(甚至是整个系统中)其他模块或变量、数组等的存储空间，尽管编译时不会有任何问题，但在程序运行时操作系统一般会给出保护模式错误警告。

数组元素可以当成普通的变量使用，也就是说在程序中所有变量可以出现的地方都可以用数组元素替代，当然数组元素也可用于赋值语句的左边。

例如，声明一个长度为 20 的整型数组，并将数组中的各个元素按顺序赋予从 50 到 70 以 1 递增的数，即赋予数组的第 0 个元素的值为 50，赋予数组的第 1 个元素的值为 51，以此类推。为了能访问到数组中的所有元素，需要用一个简单的 for 循环语句，循环控制变量从 0 开始，以 1 为增量，一直递增到 19，而循环控制变量正好可以作为访问数组元素时的下标。要按顺序给数组元素赋予从 50～70 的值，只要让每个数组元素的值等于其下标值加上 50(也就是将循环控制变量加上 50)即可。相应的程序代码片段如下：

```
int nData[20];
for(int i = 0;i < 20;i++)
    nData[i] = i + 50;
```



**【例 5-1】** 假定一个班级有 20 名学生,所有学生某门课程的考试成绩保存在一个一维数组中。编写程序,找出该班学生该门课程考试的最高分,并计算平均分数和统计考试成绩不及格的学生人数。

**问题分析**:这是一个典型的顺序查找统计问题。可以通过数组与循环的配合遍历 20 个学生的成绩,实现查找最高分和统计总分和不及格人数的目的。这里的查找最高分实际上是在所有成绩中找到最大值,可以先假定第 1 个数组元素为最大值,之后通过循环结构将每一个数组元素与当前的最大值进行比较,并记录最大值所在位置,最终查找到最高分。程序代码如下:

```
#include < iostream >
#include < iomanip >
using namespace std;
int main()
{
    int nScore[20] = {90,88,45,92,76,59,89,93,60,51,     //定义一维数组,并赋初值
                      91,65,82,74,92,35,66,78,62,91};
    int nSum = 0,nUnPassedCount = 0;                      //计数变量清零
    //输出数组
    for(int i = 0;i < 20;i++)
    {
        if(i % 10 == 0)                                   //每输出 10 个数据后换行
            cout << endl;
        cout << setw(4)<< nScore[i];
    }
    //顺序查找和统计
    int max = nScore[0],k;                                //先假定第 1 个数组元素为最大值
    for(i = 0;i < 20;i++)
    {
        if(max < nScore[i])                               //找到最大值
        {
            max = nScore[i];
            k = i;
        }
        nSum += nScore[i];                                //求和
        if(nScore[i]< 60)                                 //统计不及格人数
            nUnPassedCount++;
    }
    //输出查找和统计结果
    cout <<"\\n 考试最高分为:"<< max <<"是第"<< k + 1 <<"个"<< endl;
    cout <<"平均分数为:"<<(float)nSum/20 << endl;   //为更精确显示结果,将 int 型转换成 float 型
    cout <<"不及格人数为:"<< nUnPassedCount << endl;
    return 0;
}
```

程序运行结果:

```
考试最高分为:93 是第 8 个
平均分数为:73.95
不及格人数为:4
```

程序中首先定义了一个长度为 20 的一维数组,并将 20 名学生的考试成绩赋值给一维数组。接着的第 1 个 for 语句按每行 10 个数据的格式输出 20 个考试成绩,此功能是通过选择语句判断如果循环变量 i 能被 10 整除则回车换行实现的。接着假定第 1 个数组元素 nScore[0]为最大值并赋值给变量 max,第 2 个 for 语句控制循环 20 次,通过遍历每一个学生的考试成绩,通过将数组元素 nScore[i]与 max 进行比较,并用变量 k 记录最大值的位置,最终查找出成绩最高分并存入变量 max 中,在此遍历学生考试成绩的同时,判断每一个学生的考试成绩是否小于 60,据此统计不及格人数。

**【例 5-2】** 输入 3 个不同的整数,找出最大的数和中间的数,并输出结果。

**问题分析**:如果不使用数组,在 3 个数中找最大的数和中间的数是比较麻烦的。现在将 3 个数赋值给一维数组,并与循环控制结构结合将数组中所有元素重新按照递增顺序排序,这样在排序后的数组中就能很容易地找到最大的数和中间的数。程序代码如下:

```
#include <iostream>
using namespace std;
const int N = 3;
int main()
{
    int a[N];
    int i,j,t,k;

    cout <<"输入任意 3 个不同的数:";
    for(i = 0;i < N;i++)                          //循环输入 N 个值,并赋值给一维数组 a[]
        cin >> a[i];

    for(i = 0;i < N - 1;i++)                      //或 for(i = 0;i < 2;i++)
    {
        k = i;
        for(j = i + 1;j < N;j++)                  //或 for(j = i + 1;j < 3;j++)
            if(a[j]< a[k]) k = j;
        if(k!= i)
        {
            t = a[i];
            a[i] = a[k];
            a[k] = t;
        }
    }
    cout <<"按照递增排序后的数组为:";
    for(i = 0;i < N;i++)
        cout << a[i]<<"\\t";

    cout <<"\\n3 个数中最大值为:"<< a[2]<< endl;
    cout <<"3 个数中中间的值为:"<< a[1]<< endl;

    return 0;
```

```
}
```

程序运行结果：



```
输入任意3个不同的数:7  3  8
按照递增排序后的数组为:3  7  8
3个数中最大值为:8
3个数中中间的值为:7
```

## 5.3 多维数组

如果一个数组元素本身也是数组，就形成多维数组。一维数组的下标数只有一个，而多维数组的下标数有多个，多维数组中最常用的是二维数组。

### 5.3.1 二维数组的声明

声明一个二维数组的形式为：

**<类型标识符><数组名>[第1维长度][第2维长度]**

同一维数组一样，二维数组的数组名必须遵循C++标识符的命名规则。数组第1维长度和第2维长度都是常量表达式，常量表达式中同样不能有任何变量出现。二维数组通常用于存放矩阵或二维表中的数据，因此在二维数组中，常称第1维为行，第2维为列。

例如，语句 int nMatrix[3][4]将声明一个名为 nMatrix 且第1维长度为3、第2维长度为4的二维数组，称为3行4列的二维数组，与二维表格之间的对应关系如表5-2所示。

**表5-2 二维数组与二维表格之间的对应关系**

| 行＼列 | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | nMatrix[0][0] | nMatrix[0][1] | nMatrix[0][2] | nMatrix[0][3] |
| 1 | nMatrix[1][0] | nMatrix[1][1] | nMatrix[1][2] | nMatrix[1][3] |
| 2 | nMatrix[2][0] | nMatrix[2][1] | nMatrix[2][2] | nMatrix[2][3] |

由表5-2可知，对于第1行，第1维的值都为0，且从左到右第2维的值由0增至3；在第2行，第1维的值都为1，从左到右第2维的值重新由0增至3；在第3行，第1维的值都为2，从左到右第2维的值再次由0增至3。因此，如果将二维数组名和第1维下标看作一个整体当作一维数组名，则二维数组中每行的元素就相当于一个一维数组。例如，如果将nMatrix[0]看作一个数组名，则第一行的4个元素就组成了一个长度为4的一维数组。那么3行4列的二维数组可以看成由3个长度分别为4的一维数组所构成，这3个一维数组名分别为 nMatrix[0]、nMatrix[1] 和 nMatrix[2]。

不难算出，3行4列的二维数组中共有3×4＝12个整型元素，这12个元素在内存中是按顺序存放的，先从左到右存放第一行的4个元素，紧接着从第二行的第1个元素开始，从左到右依次存放第二行的4个元素，最后存放第三行的4个元素。了解二维数组元素在内存中的存储方式，对在程序中访问二维数组元素很有益处。

### 5.3.2 二维数组的初始化

同一维数组一样，二维数组也可以在声明时赋初始值，形式如下。

形式 1：

```
<类型标识符><数组名>[第 1 维长度][第 2 维长度] = {{第 0 个第 2 维数据组},
                                                  {第 1 个第 2 维数据组},
                                                  … ,
                                                  { 第 n - 1 个第 2 维数据组}}
```

其中，n 等于第 1 维长度。

形式 2：

```
<类型标识符><数组名>[第 1 维长度][第 2 维长度] = {第 0 个元素值,
                                                  第 1 个元素值,
                                                  … ,
                                                  第 m - 1 个元素值}
```

其中，m 等于第 1 维长度乘第 2 维长度之积。

这两种形式中，如果花括号中给出的元素个数少于实际的元素个数，则剩余的数组元素将会自动赋值为 0，也称“清零”；如果花括号中给出的元素个数大于实际的元素个数，则编译器会给出错误信息。例如：

```
int a[ ][3] = {{1},{2,3},{4,5,6},{0}};          //合法,a[0][0] = 1,a[1][0] = 2, …
int b[ ][3] = {{,1},{2,3},{4,5,6},{0}};         //不合法,因为 1 之前不能有"空"
int c[ ][3] = {{1},{2,3},{4,,6},{0}};           //不合法,因为 4 和 6 之间不能有"空"
int d[2][2] = {1,2,3,4,5}                       //不合法,初值个数大于数组的元素个数
```

### 5.3.3 访问二维数组的元素

要访问二维数组中的元素，同样需要指定要访问的元素的下标。二维数组的元素有两个下标，访问二维数组元素的形式为：

```
<数组名>[第 1 维下标][第 2 维下标]
```

这里，下标的值也是从 0 开始，且不能超过该维的长度减 1。下标可以是任意整型表达式，只要其值在该下标的有效范围内即可。

要访问二维数组中的某个元素，必须给出该元素所在的行和列。如 nMatrix[2][1]代表数组名为 nMatrix 的二维数组中位于第 3 行、第 2 列的元素。同一维数组一样，二维数组的元素也可以当成变量进行赋值或参与各种表达式的计算。

**【例 5-3】** 生成如下格式的方阵，将其存入二维数组中。

(1) 输出二维数组所有元素的值。

(2) 求该方阵每行、每列及对角线(左上右下)之和。

```
1   2   3   4   5
10  9   8   7   6
11  12  13  14  15
```

```
20  19  18  17  16
21  22  23  24  25
```

**问题分析**：二维数组的行和列同矩阵的行和列是对应的。因此，为了访问二维数组中的所有元素，应使用双重循环，外层循环控制变量作为当前行，内层循环的循环控制变量作为当前列。

这个方阵的规律在于：偶数行中的元素按升序排列，奇数行中的元素按降序排列，只要按此规律逐行“处理”方阵中的元素，即可得此方阵。在显示二维数组时，为了得到理想的显示效果，要对不同的元素指定不同的显示位置。另外，考虑除需要存放5×5方阵的各元素，还要存放各行和各列以及左上右下对角线之和，因此定义一个6×6的方阵，最后一列数组元素用于存放对应行之和，最后一行的数组元素用于存放对应列之和，最后一个数组元素则用于存放左上右下对角线上各数之和。程序代码如下：

```
#include <iomanip>
#include <iostream>
using namespace std;
int main()
{
    int nRow;                                   //控制行的变量
    int nCol;                                   //控制列的变量
    int nMatrix[6][6] = {0};                    //二维数组声明,且数组元素被赋值为0
    //生成5×5方阵
    for(nRow = 0;nRow < 5;nRow++)
    {
        for(nCol = 0;nCol < 5;nCol++)
        {
            if(nRow % 2 == 0)
                nMatrix[nRow][nCol] = nRow * 5 + nCol + 1;
            else
                nMatrix[nRow][4 - nCol] = nRow * 5 + nCol + 1;
        }
    }
    //输出方阵
    cout <<"生成方阵为: \n";
    for(nRow = 0;nRow < 5;nRow++)
    {
        for(nCol = 0;nCol < 5;nCol++)
        {
            cout << nMatrix[nRow][nCol];
            if(nMatrix[nRow][nCol]< 10)         //控制输出1位数与2位数时的不同间隔
                cout << setw(3)<<"  ";
            else
                cout << setw(2)<<"  ";
        }
        cout << endl;                           //每输出一行后换行
    }
    cout << endl;
    //计算5×5方阵各行及左上右下对角线之和
    for(nRow = 0;nRow < 5;nRow++)
```

```
    {
        for(nCol = 0;nCol < 5;nCol++)
        {
            nMatrix[nRow][5] += nMatrix[nRow][nCol];                    //计算各行之和
            nMatrix[5][nRow] += nMatrix[nCol][nRow];                    //计算各列之和
        }
        nMatrix[5][5] += nMatrix[nRow][nRow];
        cout <<"第"<< nRow + 1 <<"行之和: "<< nMatrix[nRow][5]<<"\t\t";
        cout <<"第"<< nRow + 1 <<"列之和: "<< nMatrix[5][nRow]<< endl;
    }
    cout <<"\n 左上右下对角线之和: "<< nMatrix[5][5]<< endl;
    return 0;
}
```

程序运行结果：

```
生成方阵为:
    1   2   3   4   5
    10  9   8   7   6
    11  12  13  14  15
    20  19  18  17  16
    21  22  23  24  25

第 1 行之和: 15          第 1 列之和: 63
第 2 行之和: 40          第 2 列之和: 64
第 3 行之和: 65          第 3 列之和: 65
第 4 行之和: 90          第 4 列之和: 66
第 5 行之和: 115         第 5 列之和: 67

左上右下对角线之和: 65
```

特别需要说明的是，语句

```
int nMatrix[6][6] = {0};
```

与语句块

```
int nMatrix[6][6];
for(nRow = 0;nRow < 6;nRow++)
    for(nCol = 0;nCol < 6;nCol++)
        nMatrix[nRow][nCol] = 0;
```

的作用相同，都是将数组元素“清零”。前者是在声明二维数组的同时，只将数组元素nMatrix[0][0]赋值为0，剩余的数组元素将自动“清零”；后者是在二维数组声明后，通过循环逐个将每一个数组元素赋值为0。

如果数组只声明不赋值，其数组元素不会自动“清零”，这时数组元素的值是不确定的，不能直接使用。如果将数组用 static 说明为静态的，则数组中所有元素将自动赋初值为0，无须再对数组进行“清零”的操作，即

```
static int nMatrix[6][6];
```

可以替代以下代码：

```
int nMatrix[6][6];
for(nRow = 0;nRow < 6;nRow++)
    for(nCol = 0;nCol < 6;nCol++)
        nMatrix[nRow][nCol] = 0;
```



## 5.4 数组作为函数参数

C++将数组名解释为该数组第一个元素的地址，并视数组名为指针。有关指针的概念将在第7章介绍，这里只结合数组参数的传递过程，对数组名作为函数参数的编程方法进行介绍。

### 5.4.1 一维数组名作为参数

在涉及函数调用的程序设计中，通常需要将数组中存放的所有数据传递给被调用函数。最直接的想法是将该数组中的所有元素都作为参数，一个一个地传递给被调用函数。显然，随着数组元素的增多，函数的参数阵容将非常庞大，所以这种方法基本上是不可行的。在实际的程序设计中，通常采用只将数组名(即数组中第一个数组元素的地址)作为实参传递给被调用函数的方法，实现将整个数组传递给被调用函数的目的。例如，在主函数main()中声明了如下数组：

```
int nScore[20];
```

如果要将该数组中的所有数据传递给另一个函数func()，则在函数main()中，可用下列形式的函数调用语句：

```
func(nScore,20);
```

这里，数组名nScore是数组元素nScore[0]的内存地址，20是整个数组的元素个数，即数组的长度。

一维数组作为参数时，函数的声明主要有以下两种形式。

形式1：

```
<类型标识符><函数名>(类型标识符数组名[],int 长度)
```

形式2：

```
<类型标识符><函数名>(类型标识符数组名[长度])
```

第一种形式适用于处理不同长度的数组，数组的实际长度通过另一个参数传递给函数；第二种形式只可用于传递长度固定的数组。不管哪一种形式，传递的都不是数组本身，而是传递第一个数组元素所在内存单元的地址，即数组的起始地址。通过传递数组的起始地址，被调用函数可得到数组的准确存放位置。因此，当被调用函数在函数体中修改数组元素的值时，实际上是修改该地址所指的内存单元中原数组元素的值。

**【例5-4】** 一个班级有20名学生，所有学生的英语考试成绩保存在一个一维数组中。