第5章 数 组

本章学习目标

- 数组的作用。一维数组与二维数组的概念、定义，数组元素的引用、初始化、输入与输出的方法，与数组有关的程序设计算法(如冒泡排序算法)
- 一维数组与二维数组元素下标的表示方法
- 使用字符数组存储字符串、输入与输出字符串的方法以及常用的字符串处理函数

在程序设计中，经常需要对若干个具有相同数据类型的数据进行分析与处理。如果使用基本数据类型(整型、实型、字符型)变量来处理这样众多的数据，使用起来会很不方便。

例如，某个班级有 60 个学生，分别用 int 类型变量 score1，score2，…，score60 来存放每一个学生的成绩。依次输入每个学生的成绩，如果采用 C 语言顺序结构程序设计的方法，则需要书写 60 个 scanf 函数来完成。即：

```
scanf("%d",&score1);
scanf("%d",&score2);
…
scanf("%d",&score60);
```

如果在同一个 scanf 函数中完成对 60 个成绩数据的输入，仅格式控制符“%d”就需要书写 60 次。即：

```
scanf("%d,%d,.....,%d",&score1,&score2,.....,&score60);
          共 60 个                    共 60 个
```

可见工作量较大。为了解决诸如此类的复杂问题，C 语言提供了构造数据类型，即把若干个基本数据类型按照一定规则构造，主要有数组类型、结构体类型、共同体类型等。本章介绍 C 语言中最常用的一种构造数据类型——数组类型。数组是由相同数据类型的元素组成的数据集合，也是若干个同类型变量的有序集合，这些元素存放在计算机内存中一个连续的存储区域内。数组用统一的数组名和不同的下标来唯一标识数组中的每一个元素。

同样是依次输入 60 个学生的成绩，如果使用数组来解决，即程序段书写形式如下：

```
…
for(i=0;i<60;i++)
scanf("%d",&score[i]);        //依次完成对 60 个成绩数据的输入;
…
```

显而易见，采用数组处理多个相同类型的数据时，会大大简化书写程序代码的工作量，使程序简明而高效。

5.1 一维数组

一维数组用一维线性顺序关系把若干个具有相同数据类型的数据组织起来，这些数据

在计算机内存中占有连续的存储空间。一维数组较简单，只需要用数组名与一个下标就能够唯一确定数组中的元素。

5.1.1 一维数组的定义

与C语言基本类型的数据一样，数组也必须先定义，后使用。即事先“告诉”计算机由哪些数据组成数组，这些数据属于什么样的数据类型，数组中允许最多具有这些数据元素的个数。

1. 定义形式

一维数组的定义形式如下：

```
类型说明符 数组名[常量];
```

例如：

```
int b[5];float a_1[10];char abc[4];
```

说明：

① 数组名的命名规则要遵循C语言标识符的命名规则，且数组名不能与同一程序中其他的变量名相同。例如，在某C程序段中：

```
main()
{
    int b;
    int b[5];          //非法，数组名b与已有变量重名;
    …
}
```

对数组b的定义是错误的，因为数组名“b”与普通变量b的名字重合了。

② 方括号[]表明定义的是数组变量，不能用其他样式的括号，如{ }、()、< >等。定义数组时，方括号中的正整数表明了该数组中所能容纳(包含)元素的个数，也称之为该数组的长度。数组中所含元素的个数不能用小数、0、负数等来表示。

例如，对数组的定义：

```
float a_1[10];
```

则表明数组a_1中最多可以容纳10个元素。

如果写成float a_1[9.5];float a_1[0];float a_1[-10];float a_1[];等形式，均是错误的。

③ 数组的类型其实就是数组中所含元素的数据类型。对于同一个数组，其所含元素的数据类型都是相同的。

④ 定义数组时，数组中所含元素的个数不能用变量表示。

例如：

```
int n=4;
char abc[n];          //n是变量,非法!
```

但是，允许使用整型常量表达式或符号常量来表示数组中所含元素的个数。（本书建议初学者不要采用这种方式。）

例如：

```
int b[3+2];    //"3+2"是整型常量表达式,合法!
```

⑤ C 语言允许对具有同一数据类型的多个数组以及多个变量同时定义。

例如：

```
int x,y,z,a[5],b[8];
```

2. 数组内容及其存储结构

在计算机内存里，数组中的元素是按照由低地址至高地址的顺序依次存放的，存放次序不能颠倒。每一个数组元素所占内存空间大小由该数组的数据类型所占内存空间大小决定。

例如：

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

定义了一个 a 数组，类型为 int 类型。它含有 5 个元素，依次表示为 a[0]、a[1]、a[2]、a[3]、a[4]。数组 a 在计算机内存里的存储结构如图 5.1 所示。其中，每一个元素所占内存空间大小为 4 个字节。

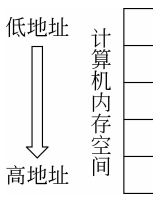


图 5.1　一维数组 a 在计算机内存里的存储结构

需要注意的是，一旦定义了某一个数组，其数组名中存放的是一个地址常量，它代表了该数组的首地址（有关数组的首地址及其应用，第 7 章将介绍），也就是数组中第一个元素的地址。

比如，在 a 数组中，元素 1(a[0])的地址就是 a(a 数组的数组名)。而其余元素的地址则为数组首地址加上该元素位置偏移量(该元素所在位置减去第一个元素位置)与数组类型所占内存字节数的乘积。即：

元素 2(a[1])的地址：a＋1＊4(字节)＝a＋4

元素 3(a[2])的地址：a＋2＊4(字节)＝a＋8

元素 4(a[3])的地址：a＋3＊4(字节)＝a＋12

元素 5(a[4])的地址：a＋4＊4(字节)＝a＋16

5.1.2　一维数组元素的引用

定义完一个数组之后，就可以引用数组中的任意一个元素。引用一维数组元素的表示形式如下：

```
数组名[下标表达式];
```

有几点说明如下：

① 不能对数组进行整体引用，也不能整体引用数组中的全部元素，只能逐个引用数组中的每一个元素。数组中的一个元素，其实也就是一个简单变量，可以对它进行赋值操作以

及做各种数学运算等，具有与同一数据类型的其他简单变量同样的属性。

② “下标表达式”可以是整型常量、整型变量或者是一个返回整型量的表达式。但是，“下标表达式”的值必须是非负整数。

③ 对于已定义的数组，在引用数组元素时，其下标表达式的取值范围是 0 至“数组中所能包含元素的个数－1”。也就是说，对于该数组中的第一个元素，引用时的下标值为 0，而不是 1；而对于该数组中的最后一个元素，引用时的下标值为 n－1，而不是 n（假设 n 为数组中所能包含元素的个数，即数组长度）。

④ 不能把数组当做一个整体参与数学运算，但是可以对数组中的任意一个元素进行相应的数学运算。

例 5.1 假设定义数组 a：int a[5]；下列对数组 a 中元素的引用方式，哪些是正确的？

① a[0]；② a[5]；③ a[－1]；④ a[1]＋ a[3]；⑤ a[2]＊5；⑥ 1＋a；

分析：①正确，a[0]表示引用数组 a 中的第 1 个元素。②③错误，对于 a[5]与 a[－1]，引用时的下标值不合法。④⑤正确，可以对数组中的元素进行相应的数学运算。a[1]＋a[3]；表示计算 a 数组中第 2 个元素与第 4 个元素之和；a[2]＊6；表示 a 数组中第 3 个元素乘以 5。⑥错误，不能把数组 a 当做一个整体参与数学运算。

例 5.2 依次输入 5 个整数，要求将这 5 个整数顺序输出与逆序输出。

分析：定义一个 int 类型的数组 a，数组长度为 5。可以通过 for 语句依次输入 5 个整数，并完成顺序与逆序输出。

C 程序代码如下：

```
#include <stdio.h>
main()
{
    int i,a[5];
    printf("依次输入 5 个整数: \n");
    for (i=0;i<5;i++)
    {
        scanf("%d",&a[i]);
    }
    for (i=4;i>=0;i--)
    {
        printf("%d\t",a[i]);
    }
    printf("\n");
}
```

假设依次输入：

```
5  4  6  7  1↙
```

程序运行结果如下：

```
依次输入5个整数:
5 4 6 7 1
1       7       6       4       5
Press any key to continue
```

5.1.3 一维数组元素的初始化

数组元素的初始化，也就是在定义数组时为数组中的元素赋初值。一维数组元素初始化的形式如下：

```
类型说明符 数组名[常量]={初始值表};
```

数组元素的初始值写在一对大括号{ }里面，每个初始值之间要用逗号，分开。

在C语言中，对一维数组元素的初始化分为全部元素初始化与部分元素初始化两种情形。

1. 全部元素初始化

定义一维数组时，对数组中所有的元素都赋初值。

例如：

```
int a[5]={ 3,4,5,6,7};
```

在数组a中，其所含5个元素的初始值分别为3、4、5、6、7。即a[0]=3、a[1]=4、a[2]=5、a[3]=6、a[4]=7。

说明：

① 对一维数组进行全部元素初始化时，数组中所含元素的个数(数组长度)要与元素初始值的个数相等。

比如，下面的两种写法均是错误的：

```
int a[4]={ 3,4,5,6,7};
```

错误原因：数组a的长度(4)小于元素初始值的个数(5)。

```
int a[6]={ 3,4,5,6,7};
```

错误原因：数组a的长度(6)大于元素初始值的个数(5)。尽管C编译系统不会报错，但是，这已不是对数组a中的全部元素进行初始化了。

② 对一维数组进行全部元素初始化时，一维数组的长度值在定义时可以省略。

例如：

```
int a[ ]={ 3,4,5,6,7};
```

等价于：

```
int a[5]={ 3,4,5,6,7};
```

2. 部分元素初始化

定义一维数组时，只对该数组中的部分元素赋初值。对于未被赋初值的元素，系统为其自动赋以“0”值(整型与实型)或‘\0’(字符型)。

例如：

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

在数组a中，只对其中的前3个元素赋了初值，分别为2、4、5。即a[0]=2;a[1]=4;

a[2]=5。而后 2 个元素的初值则默认为 0,即 a[3]=0;a[4]=0。

说明:

① 对一维数组中的部分元素初始化时,C 编译系统将按照对数组元素的前后顺序进行赋值操作。

例如:

```
int b[5]={ 1,4};
```

即"1"与"4"分别为数组 b 中前 2 个元素的初值。(数组 b 中后 3 个元素的初值均为"0")

② 对一维数组中的部分元素初始化时,数组的长度值定义时不能省略。

注:对于只定义而未初始化的数组,如果数组的存储类别是静态存储类别或外部存储类别,系统为数组中的所有元素自动赋上数值"0"(整型与实型数组)或'\0'(字符型数组)。如果是自动存储类别与内部存储类别的数组,则数组中所有元素的初始值是不确定的。(有关 C 语言数据的存储类别,本书将在函数一章中详细介绍。)

5.1.4 一维数组应用举例

例 5.3 从键盘上任意输入 6 个整数,要求输出其中的最大值。

分析:定义一个数组长度为 6 的整型数组 a,用来存放输入的 6 个整数。比较前,先假设数组 a 中的第一个元素 a[0]为最大值,然后与数组 a 中的第二个元素 a[1]作比较。如果 a[0]大于 a[1],则最大值仍为第一个元素 a[0];反之,最大值则变成了第二个元素 a[1]。用二者的最大值再与第三个元素 a[2]作比较,用同样的方法求得最大值。以此类推,直至比较出数组 a 中 6 个元素的最大值。这种比较方法也被称做"打擂台"的方法,适合求解数组元素的最值问题。

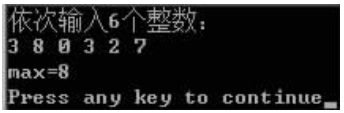
C 程序代码如下:

```
#include <stdio.h >
main()
{
    int i,a[6],max;
    printf("依次输入 6 个整数: \n");
    max=a[0];
    for (i=1;i<6;i++)
    {
        scanf("%d",&a[i]);
        if(a[i]>=max) max=a[i];
    }
    printf("max=%d \n",max);
}
```

假设依次输入:

```
3  8  0  3  2  7↙
```

程序运行结果如下:



例 5.4 任意输入 8 个整数,要求按照由小到大的顺序排序并输出。

分析:在数学中,完成对 n 个数的排序问题可以有很多种方法。如冒泡排序、快速排序、希尔排序等。这里主要介绍通过“冒泡排序”的方法完成排序过程。

冒泡排序的算法思想如下:对于任意 n 个数,从左至右,通过相邻两个数之间的比较和交换,始终保持数值较小的数在前,数值较大的数在后。每一轮比较结束后,即最大的数被交换到该轮所有数字的最后(最右端)。如此类推,除去第一轮比较结束后所得到的最大数(已位于所有数字的最右端),从最左边第一个数开始,用同样的方法对剩下的 n−1 个数进行排序与排序,完毕后,则所有数中第二大的数又被交换到右端倒数第二的位置(位于最大数的左边)。这样,经过了 n−1 轮排序之后,便完成了对 n 个数由小至大的排序过程。

以 6 个数(2、7、5、6、8、1)为例,进行冒泡排序:

第一轮排序:2 7 5 6 8 1

第一次 2 和 7 比较,不交换:2 7 5 6 8 1

第二次 7 和 5 比较,交换: 2 5 7 6 8 1

第三次 7 和 6 比较,交换: 2 5 6 7 8 1

第四次 7 和 8 比较,不交换:2 5 6 7 8 1

第五次 8 和 1 比较,交换: 2 5 6 7 1 8

在第一轮排序中,6 个数比较了 5 次,得到排序序列:2 5 6 7 1 **8**

可见,6 个数中的最大数“8”排到了最后。

第二轮排序:2 5 6 7 1 **8**(其中“8”不参与比较)

第一次 2 和 5 比较,不交换:2 5 6 7 1 8

第二次 5 和 6 比较,不交换:2 5 6 7 1 8

第三次 6 和 7 比较,不交换:2 5 6 7 1 8

第四次 7 和 1 比较,交换: 2 5 6 1 7 8

在第二趟排序中,最大数“8”不参与比较,其余的 5 个数比较了 4 次,得到排序序列:2 5 6 1 **7 8**

可见,6 个数中的第二大数“7”排到了右端倒数第二的位置(“8”的左边)。

以此类推……

第三轮排序,比较 3 次,排出:2 5 1 **6 7 8**

第四轮排序,比较 2 次,排出:2 1 **5 6 7 8**

第五轮排序,比较 1 次,排出:1 **2 5 6 7 8**

最后还剩下 1 个数“1”,其实不需再比较,写在最左端,而得到最终的排序结果:**1 2 5 6 7 8**

下面是采用冒泡排序的算法思想,实现对任意 8 个数由小到大排序的实现程序。

C 程序代码如下:

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

```
main()
{
    int i,j,temp,a[8];
    for(i=0;i<8;i++)
    scanf("%d",&a[i]);
    for(i=0;i<7;i++)                    //外层循环控制比较轮数;
    {
        for(j=0;j<8-(i+1);j++)          //内层循环控制在每轮中,数字比较的次数;
        {
            if(a[j]>=a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
        for(i=0;i<8;i++)                //排序结果的输出;
        printf("%d\t",a[i]);
}
```

假设依次输入：

```
7  6  12  3  1  5  0  2↙
```

程序运行结果如下：

```
依次输入8个整数:
7 6 12 3 1 5 0 2
排序后的结果:
0       1       2       3       5       6       7       12
Press any key to continue
```

实训 8　一维数组应用实训

任务 1　任意输入 6 个整数，求出前三个数的最大值与后三个数的平均值（结果保留两位小数）。

C 程序代码如下：

```
#include <stdio.h>
main()
{
    int i,a[6],max,s=0;
    float ave;
    printf("依次输入 6 个整数: \n");
    for(i=0;i<6;i++)
    {
        scanf("%d",&a[i]);
    }
```

```
    max=a[0];
    for(i=1;i<3;i++)
    {
        if(max<=a[i]) max=a[i];
    }
    printf("max=%d\n,max");
    for(i=3;i<6;i++)
    {
        s=s+a[i];
    }
    ave=s/3.0;
    printf("ave=%.2f\n",ave);
}
```

请读者调试程序,观察与分析运行结果。

假设依次输入:

```
4  7  9  1  8  4↙
```

程序运行结果如下:

```
依次输入6个整数:
4 7 9 1 8 4
max=9
ave=4.33
Press any key to continue
```

任务 2 任意输入 10 个整数,统计出正数、负数与零的个数,并计算出所有的正数之和与负数之和。

C 程序代码如下:

```
#include <stdio.h >
main()
{
    int a[10],zhengshu=0,fushu=0,zero=0,sumz=0,sumf=0,i;
    //变量 zhengshu、fushu、zero 分别表示正数、负数、零的个数,其初始值为 0; 变量 sumz、
    sumf 分别表示所有的正数之和以及负数之和,其初始值为 0;
    printf("依次输入 10 个整数: \n");
    for(i=0;i<10;i++)
    {
        scanf("%d",&a[i]);
    }
    for(i=0;i<10;i++)
    {
        if(a[i]>0)
        {
            sumz=sumz+a[i];zhengshu++;
        }
        else if(a[i]<0)
```

```
        {
            sumf=sumf+a[i];
            fushu++;
        }
        else
        zero++;
    }
    printf("sumz=%d,sumf=%d\n",sumz,sumf);
    printf("zhengshu=%d,zero=%d,fushu=%d\n",zhengshu,zero,fushu);
}
```

请读者调试程序，观察与分析运行结果。

假设依次输入：

```
3  4  1  7  -8  -1  0  -4  9  5↙
```

程序运行结果如下：

```
依次输入10个整数:
3 4 1 7 -8 -1 0 -4 9 5
sumz=29,sumf=-13
zhengshu=6,zero=1,fushu=3
Press any key to continue
```

任务 3　已知 int 型数组 a[7]，数组中包含的前 6 个元素已按照由小至大的顺序排好次序，分别为 1、3、5、8、9、13。从键盘上任意输入一个整数并插入到数组 a 中，要求插入新数之后数组元素 a 中的 7 个元素仍然有序排序。

分析：该问题具有一定的难度。在初始状态下，原数组内部的 6 个元素已排好顺序，我们采取这样一个思路：从数组中的第一个数开始，依次与新(插入的)数进行比较，在原数组中找到合适的插入(新数)位置。然后将该位置原有的数以及其后面的数依次“后移”一个位置，即为新插入的数“空出”一个位置。当然，也会存在一种特殊情况，即新插入的数值比原数组中的最后一个数(a[5])都要大(例如，插入数值 14)，这时只要把该数放至数组中的最后一个位置上，而其余数组元素则不必移动。

所以，按照上述思路，给出了实现任务 3 的 C 程序，仅供参考。

C 程序代码如下：

```
#include <stdio.h>
main()
{
    int a[7]={1,3,5,8,9,13};
    int t1,t2,b,end,i,j;
    printf("原数组元素队列: \n");
    for(i=0;i<6;i++)
    printf("%4d",a[i]);
    printf("\n");
    printf("插入一个新的数: \n");
```