第5章 数 组

在程序设计中，为了处理方便，把具有相同类型的若干变量按有序的形式组织起来。这些按序排列的同类数据元素的集合称为数组。在C语言中，数组属于构造数据类型。一个数组可以分解为多个数组元素，这些数组元素可以是基本数据类型或是构造类型。因此按数组元素的类型不同，数组又可分为数值数组、字符数组、指针数组和结构数组等各种类别。难点是对数组名含义的理解和字符串的相关操作。

5.1 数组引例

【例5-1】 编写程序计算10位学生的C语言的平均成绩，并统计比平均分高的人数。

分析：显而易见，该程序是例4-2的深入。例4-2求的是一个班级内学生的总成绩，所以每位同学的成绩并不需要存放。程序内使用score每录入一个新的学生成绩，统计入总分sum后，下一位同学的成绩均会在score内将上一人成绩覆盖。而此处学生的成绩需要保留，就不能只用一个变量，而需要定义10个变量，例如：

```
int   n, s, score1, score2, score3, score4, score5, score6, score7, score8, score9,
score10;
float  aver;
scanf( "%d%d%d%d%d%d%d%d%d%d", &score1, &score2, &score3, &score4, &score5,
&score6,&score7,&score8,&score9,&score10);
s=score1+score2+score3+score4+score5+score6+score7+score8+score9+score10;
aver=s/10;
    if(score1>aver)   printf("%d",score1);
    if(score2>aver)   printf("%d",score2);
    if(score3>aver)   printf("%d",score3);
…
```

这样的定义方法显然有违常理，工作量过大。如果不是10个数，而是100，1000，甚至是10000，此时按上面方法编写程序就非常冗长。

从而引入数组，int score[10]即可一次性定义10个变量。同时，如果可以使用循环来编写，程序可以简洁许多。要使用循环：必须使用score[i](i=1,2,…,10)的形式来代表score1，score2，…，score10，则程序编写修改为如下形式：

```
#include<stdio.h>
void main()
{
    float score[10],aver,sum=0.0;
    int i,num=0;
    for(i=0;i<=9;i++)
```

```
    {
        printf("Input students %i score:",i+1);
        scanf("%f",&score[i]);
        sum=sum+score[i];
    }
    aver=sum/10;
    for(i=0;i<=9;i++)
        if(score[i]>aver)    num=num+1;
    printf("平均分为%f,高于平均分人数为%d\n",aver,num);
}
```

程序运行结果：

```
Input students 1 score: 10
Input students 2 score: 20
Input students 3 score: 30
Input students 4 score: 40
Input students 5 score: 50
Input students 6 score: 60
Input students 7 score: 70
Input students 8 score: 80
Input students 9 score: 90
Input students 10 score: 100
```

平均分为 55.000000，高于平均分人数为 5。

题目中利用了变量 i 作为统计数据个数的下标，所以在 printf("Input students %i score:",i+1);语句中，i 初值为 0，i+1 体现了学生的序号。而 for 循环中的 score[i]则分别在每次循环内代表 score[0]，score[1]，score[2]，…，score[9]这十个变量记录的十位同学的各自成绩。

数组的学习当中，一定要会使用循环控制变量 i 作为数组下标的使用方法。

数组是一些具有相同类型的数据的集合，是属于构造类型（又称导出类型）的数据。同一个数组中的每个元素都具有相同的变量名，但具有不同的序号，也即下标。

一维数组：只有一个下标的数组。

二维数组：有两个下标的数组。

以此类推，C 语言允许使用任意维数的数组。当处理大量的和同类型的数据时，利用数组是很方便的。数组同其他类型的变量一样，也必须先定义，后使用。

在程序中使用数组，一是刻意用数组名加下标来表示同类型的数据，不需定义多个变量；二是同类型的数据在内存中连续存放，便于实现对数组的高效管理。

5.2 一维数组

只有一个下标变量的数组，称为一维数组。定义一个一维数组，需要明确数组名，数组元素的数据类型和数组中不包含的数组元素个数（即数组长度）。

5.2.1 一维数组定义

一般形式为：

```
类型符  数组名[常量表达式];
```

其中：

类型说明符是任一种基本数据类型或构造类数据类型；

数组名是用户定义的标识符；

方括号中的常量表达式表示数据元素的个数，也称为数组的长度。

例如：

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

表示定义了 5 个连续的存储空间分别存放 a[0]，a[1]，a[2]，a[3]，a[4]这 5 个变量，其中注意下标从 0 开始，并且每个元素所能存放的变量类型为 int 整型，其在内存中存放形式为：

a[0]	a[1]	a[2]	a[3]	a[4]

在定义数组时，需要注意如下几个问题：

(1) 表示数组长度的常量表达式，必须是正的整型常量表达式，通常是一个整型常量。

(2) C 语言不允许定义动态数组，也就是说，数组的长度不能依赖于程序运行过程中变化着的变量。下面这种数组定义方式是不允许的：

```
int i;
i=15;
int data[i];
```

(3) 相同类型的数组、变量可以在一个类型说明符下一起说明，互相之间用逗号隔开。例如：

```
float a[10],f,b[20];
```

它定义了 a 是具有 10 个元素的浮点型数组，f 是一个浮点型变量，b 是具有 20 个元素的浮点型数组。但数组名不能与同一函数中其他变量名相同：int a；float a[10]；是错误的。

5.2.2 一维数组引用和初始化

1. 数组元素的引用

一维数组元素的表示方法如下所示：

```
数组名[下标表达式]
```

其中，下标表达式可以是整型常量、整型变量及其表达式。当数组的长度为 n 时，下标表达式的取值范围为 0，1，2，…，$n-1$，也就是说，C 语言数组中的各元素总是从 0 开始编号的，即数组元素的下标是从 0 开始的。例如：

```
int a[5];          /*此处为数组的定义，表明 a 数组共有 a[0]~a[4]五个整型变量*/
```

```
a[0]=3;          /*数组元素的引用,给数组元素 a[0]赋值为 3*/
a[4]=9;          /*数组元素的引用,给数组元素 a[4]赋值为 9*/
```

注意:

(1) 引用数组元素的时候,可以用变量。以下 n 作为数组元素引用下标时可以使用变量(数组定义时不可使用变量)。以下语句均正确。

```
int n=3,a[10];
a[n]=5;
a[n+1]=10;
a[6]=a[2]+a[3]-a[2*4];
```

(2) 数组下标从 0 开始,最大下标为 9,没有 a[10]元素,以下语句错误。

```
int a[10],x,y;
a[10]=5;
```

(3) 只能逐个引用数组元素,不能一次引用整个数组。以下语句错误,不能用一个语句输出整个数组。

```
int a[10];
printf("%d",a);
```

2. 一维数组的初始化

数组一般在声明时初始化。在编译阶段进行初始化这样将减少运行时间,提高效率。数组初始化的一般形式为:

```
类型符  数组名[常量表达式]={初始化列表};
```

例如:

```
int a[10]={ 0,1,2,3,4,5,6,7,8,9 };
```

相当于

```
a[0]=0; a[1]=1;…;a[9]=9;
```

对数组元素初始化可以用以下方法实现:

(1) 对数组所有元素赋初值,可以省略数组长度。

例如:

```
int a[]={0,1,2,3,4,5,6,7,8,9};
```

(2) 也可对数组部分元素赋初值。

例如:

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

其中,a[0]~a[5]被赋初值,其余元素 a[6]~a[9]值不确定。但如果在定义数组是定义了数组的存储类型为“静态”存储,关键字为 static,则当花括号内值的个数少于数组元素的个数时,多余的数组元素初值自动为 0。例如:

```
static int a[10]={0,1,2,3,4,5};
```

则 a[6]～a[9]自动初始化初始值为 0。

(3) 给数组整体赋初值。

例如：

```
int a[10]={0};
```

此时，数组 a 的所有元素均被赋值为 0。而 int a[10]=0;是不可行的，因为不能给数组整体赋值，只能针对数组内元素操作。

(4) 循环形式为数组元素一一赋值。

例如：

```
int i,a[10];
for(i=0;i<10;i++)
    a[i]=i;
```

则与 int a[10]={ 0,1,2,3,4,5,6,7,8,9 };效果完全一致。同理，利用循环也可对数组元素任意赋值。

例如：

```
int i,a[10];
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);     /*配合 scanf()函数实现任意数值的赋值*/
```

数组元素输出同样可以利用循环的方式，这是今后数组操作最常用也是最基本的方法。

例如：

```
for(i=0;i<10;i++)
    printf("%d",a[i]);
```

5.2.3 一维数组的应用

【例 5-2】 将任意 10 个数的数组倒序输出。

分析：本题主要希望大家注意数组中下标的关系。将数组倒序输出，意味着本来 a[0]～ a[9]的输入顺序，最后要按照 a[9]～ a[0]输出。数据总个数为 $n=10$，注意输入与输出之间的对应关系，a[0]对 a[9]，a[1]对 a[8]……每对之间下标相加 $0+9=1+8=2+7=\cdots=n-1$，意味着如果当前值为 a[i]，则输出值应为 a[$n-1-$i]。

```
#include<stdio.h>
void  main()
{
    int i,a[10];
    int n=10;
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    for(i=0;i<n;i++)
```

```
        printf("%d  ",a[n-1-i]);     /*利用下标控制数组输出顺序*/
}
```

程序运行结果：

```
1 2 3 4 5 6 7 8 9 10↙
10  9  8  7  6  5  4  3  2  1
```

【例 5-3】 求 Fibonacci 数列前 20 个数。该数列的定义如下：

$$f_1 = 1, \quad n = 1$$
$$f_2 = 1, \quad n = 2$$
$$f_n = f(n-1) + f(n-2), \quad n \geqslant 3$$

分析：利用传统方法，需要找各个数据之间的关系，发现通式 $f_n = f(n-1) + f(n-2)$ $(n \geqslant 3)$是加法模式，下一个数十前两个数之和。则每次移动，此次 $f_3 = f_2 + f_1$，下次计算新数据时移动相应数据变为新加数，也即 f_2 成为新的 f_1，f_3 成为下一次计算中的 f_2，用语句 $f_1 = f_2$；$f_2 = f_3$；完成。

```
#include<stdio.h>
void main()
{
    int f1=1,f2=1,f3,count=2,i;
    printf("%5d%5d",f1,f2);
    for(i=0;i<18;i++)
    {
        f3=f2+f1;
        printf("%5d",f3);
        if(++count==5)
        {
            count=0;
            printf("\n");
        }
        f1=f2;
        f2=f3;
    }
}
```

可见，以往方法虽然也可以实现，但编程者思路必须清晰，会利用通式转换变量关系。利用数组则可以简化解题思路，直接利用下标表现通式关系即可。

```
#include<stdio.h>
void main()
{
    int i;
    int f[20]={1,1};
    for(i=2;i<20;i++)
        f[i]=f[i-2]+f[i-1];     /*直接利用下标关系表示通式，程序完成*/
    for(i=0;i<20;i++)           /*按格式输出数据*/
```

```
    {
        if(i%5==0)
            printf("\n");
        printf("%5d",f[i]);
    }
    printf("\n");
}
```

程序运行结果：

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

【例 5-4】 输入 10 个整型数据，找出其中的最大值并显示出来，并将其与第一个数组数据互换。

分析：在若干数中求最大值，一般先假设首数为最大值的初值，然后依次将每一个数与最大值比较，若大于该数，则该数替换为新的最大数。

元素互换，需要保留最大数外还需要知道其下标。利用中间变量，运用以前的知识，交换数组间的两个数。

```
#include<stdio.h>
void main()
{
    int buffer[10],Max,Max_loc,i,t;
    for(i=0;i<10;i++)
        scanf("%d",&buffer[i]);
        Max=buffer[0];       /*假设首个数为所要求的最大数*/
        for(i=1;i<10;i++)   /*遍历比较后续数据,始终保留最大数*/
            if(Max<buffer[i])
            {
                Max=buffer[i];
                Max_loc=i;
            }
    printf("Max=%d 下标为%d\n",Max,Max_loc);
    t=buffer[0];             /*数组元素的交换*/
    buffer[0]=buffer[Max_loc];
    buffer[Max_loc]=t;
    for(i=0;i<10;i++)
        printf("%d ",buffer[i]);
}
```

程序运行结果：

```
66 55 77 54 32 9 6 54 86 32↙
Max=86 下标为 8
```

86 55 77 54 32 9 6 54 66 32

【例 5-5】 选择法排序。将无序数组 8,6,9,3,2,7 排列成有序的顺序。

分析：选择法排序是最为简单且最易理解的算法，基本思想是：每次在若干个无序数中找最小(大)数，并放在无序数中第一个位置。

例对于有 n 个数的数组，按递增次序排序的步骤：

(1) 从 n 个数中找出最小数的下标，出了内循环，最小数与第 1 个数交换位置；通过这一轮排序，第 1 个数已确定好。

(2) 除已排序的数外，其余数再按步骤(1)的方法选出最小的数，与未排序数中的第 1 个数交换位置。

(3) 重复步骤(2)，最后构成递增序列。

由此可见，数组排序必须用两重循环才能实现，内循环选择最小数的下标，找到该数在数组中的有序位置；执行 $n-1$ 次外循环使 n 个数都有其确定的位置。

递减排列每次选最大数即可。

针对本题，6 个数需要经历 5 轮选择($i=0\sim4$)。

每一轮做的工作：

a. 从第 i 个到最后一个中找一个最小的。

b. 与第 i 个交换。放到 $a[i]$这个位置。

具体过程如图 5-1 所示。

						原始数据	8 6 9 3 2 7
a(1)	a(2)	a(3)	a(4)	a(5)	a(6)	第1轮比较	2 6 9 3 8 7
	a(2)	a(3)	a(4)	a(5)	a(6)	第2轮比较	2 3 9 6 8 7
		a(3)	a(4)	a(5)	a(6)	第3轮比较	2 3 6 9 8 7
			a(4)	a(5)	a(6)	第4轮比较	2 3 6 7 8 9
				a(5)	a(6)	第5轮比较	2 3 6 7 8 9

图 5-1 选择法排序过程演示图

```
#include<stdio.h>
void  main()
{
    int i,j,t,p,n=6;
    int a[6]={8,6,9,3,2,7};
    for(i=0;  i<n-1;  i++)        /*n=6个数需要经历n-1,即5轮选择(i=0~4)*/
    {
        p=i;                      /*从第i个到最后一个中找一个最小的*/
        for(j=i+1;  j<n;  j++)
            if(a[j]<a[p])
                p=j;
        t=a[i];                   /*与第i个交换*/
        a[i]=a[p];
        a[p]=t;
    }
```

```
    for(i=0;  i<n;  i++)          /*输出排序后结果*/
        printf("%d ",a[i]);
}
```

程序运行结果：

```
2 3 6 7 8 9
```

【例 5-6】 冒泡法排序。将无序数组 8,6,9,3,2,7 排列成有序的顺序。

基本思想：

(1) 从第一个元素开始，对数组中两两相邻的元素比较，将值较小的元素放在前面，值较大的元素放在后面，一轮比较比较完毕，一个最大的数沉底成为数组中的最后一个元素，一些较小的数如同气泡一样上浮一个位置。如 a[0]与 a[1]比较，a[0]>a[1]，为逆序，则两者交换；然后 a[1]与 a[2]比较……，直到最后 a[n−1]与 a[n]，此时一轮比较完成，一个最大的数沉底，成为数组最后一个数 a[n]，一些较小的数如同气泡一样上浮。

(2) n 个数，经过 n−1 轮比较后完成排序。

针对本题，具体过程如图 5-2 所示。

	第1趟 比较5次						第2趟 比较4次						第3趟 比较3次				第4趟 比较2次			比1次	
8	8	6	6	6	6	6	6	6	6	6	6	6	6	3	3	3	3	2	2	2	2
6	6	8	8	8	8	8	8	8	3	3	3	3	3	6	2	2	2	3	3	3	3
9	9	9	9	3	3	3	3	3	8	2	2	2	2	2	6	6	6	6	6	6	6
3	3	3	3	9	2	2	2	2	2	8	7	7	7	7	7	7	7	7	7	7	7
2	2	2	2	2	9	7	7	7	7	7	8	8	8	8	8	8	8	8	8	8	8
7	7	7	7	7	7	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

图 5-2　冒泡法排序过程演示图

```
#include<stdio.h>
void main()
{
    int i,j,t,n=6;
    int a[6]={8,6,9,3,2,7};
    for(i=0;  i<n-1;  i++)   /*n=6个数需要经历n-1即5轮冒泡(i=0~ 4)*/
        for(j=0;  j<n-i-1;  j++)       /*每轮需要经过5-i次比较*/
            if(a[j]>a[j+1])    /*发现前面的数比后面的数大则需要交换,把大的换到后面
                                去。即相邻逆序则交换*/
            {
                t=a[j];
                a[j]=a[j+1];
                a[j+1]=t;
            }
        for(i=0;  i<n;  i++)  /*输出排序后结果*/
            printf("%d ",a[i]);
}
```

程序运行结果：

```
2 3 6 7 8 9
```

【同步练习】

1. 利用选择法倒序输出任意 10 个数。
2. 利用冒泡法倒序输出任意 10 个数。

5.3 二维数组

5.3.1 二维数组的定义

二维数组定义的一般形式如下：

```
类型符  数组名[常量表达式 1][常量表达式 2];
```

其中：常量表达式 1 表示第一维下标的长度，常量表达式 2 表示第二维下标的长度。

例如：

```
int a[3][2];      表示 3 行 2 列的数组，共 6 个整型元素
float b[4][4];    表示 3 行 4 列的数组，共 16 个浮点元素
char c[5][10];    表示 5 行 10 列的数组，共 50 个字符元素
```

二维数组的应用与矩阵有关，其中，从左起第 1 个下标表示行数，第 2 个下标表示列数。与一维数组相似，二维数组的每个下标也是从 0 开始的。数组中的每个元素都具有相同的数据类型，且占有连续的存储空间，一维数组的元素是按照下标递增的顺序连续存放的，二维数组元素的排列顺序是按行进行的，即在内存中，先按顺序排列第 1 行的元素，然后，再按顺序排列第 2 行的元素，以此类推。例如，上面定义的 a 数组中的元素在内存中的排列顺序为：

a[0][0],a[0][1],a[1][0],a[1][1],a[2][0],a[2][1]

数组元素在内存中的排列顺序与程序设计没有直接关系，在需要时，只要利用下标来存取相应的数组元素即可。

根据 C 语言对二维数组的定义方式，可以把二维数组看成是一种特殊的一维数组：它的元素又是一个一维数组。例如，可以把数组 a 看成是一个一维数组，它有 3 个元素：a[0],a[1],a[2]，而每个元素又包含 2 个元素的一维数组，如图 5-3 所示。因此可以把 a[0],a[1],a[2]看做是 3 个一维数组的名字。上面定义的二维数组就可以理解为定义了 3 个一维数组，即相当于：

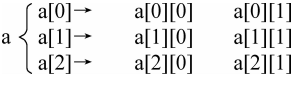

图 5-3 二维数组 a[3][2]

```
int a[0][2],a[1][2],a[2][2];
```

C 语言这种处理方法在数组初始化和用指针表示时显得特别方便，在以后学习中会体会到。按行存放二维数组的形式可以类推至多维数组，多维数组存放时各元素仍是连续的，按行存放的。