第5章 数 组

学习了C语言的基本数据类型和程序流程控制结构后，很多问题都可以描述和解决了。但是对于大规模的数据，尤其是相互间具有一定联系的数据，怎么表示和组织才能达到高效呢？C语言的数组类型为同类型数据的组织提供了一种有效的形式。

为了更好地理解数组的作用，请考虑这样一个问题：在程序中如何存储和处理具有n个整数的数列？如果n很小，比如n等于3时，显然不成问题，简单地声明3个int变量就可以了。如果n为1000，用int变量来表示这1000个数，就需要声明1000个int变量，其烦琐程度可想而知。用什么方法来处理这1000个变量呢？数组就是针对这样的问题的，它是用于存储和处理大量同类型数据的一种构造型数据类型。数组是具有一定顺序关系的同类型数据的集合，这些数据使用相同的名字（即数组名）和不同的下标进行区分，我们称之为数组元素。将数组与循环结合起来，可以有效地处理大批量的数据，大大提高了工作效率，十分方便。

字符串应用广泛，但C语言中没有专门的字符串类型，字符串是使用字符数组来存放的。同时，C语言标准函数库中提供有专门的字符串处理函数，方便对字符串的处理。

本章介绍在C语言中如何定义和使用数组，包括一维数组，二维数组，以及字符串数据的存储和处理。

学习目标：

- 掌握一维数组的定义、引用和初始化。
- 掌握二维数组的定义、引用和初始化。
- 掌握字符数组的定义、引用和初始化。
- 掌握字符串的处理方法，熟悉字符串常用处理函数。
- 熟悉数组基本操作算法，能够正确使用数组解决一般应用问题。

5.1 一维数组

5.1.1 一维数组的定义和引用

1. 一维数组的定义

在C语言中使用数组前必须先进行定义。一维数组的定义方式为：

```
类型声明符 数组名[常量表达式];
```

其中：

(1) 类型声明符是任一种基本数据类型或构造数据类型，即 int、float、char 等基本数据类型，以及第 9 章中将介绍的结构体数据类型。从这里可以看出，数组是建立在其他数据类型的基础之上的，因此数组是构造类型。

(2) 数组名是用户定义的数组标识符，命名规则遵循标识符命名规则。对于数组元素来说，它们具有一个共同的名字，即数组名。

(3) 方括号中的常量表达式表示数组元素的个数，也称为数组的长度。例如：

```
float b[10], c[20];          //定义实型数组 b,有 10 个元素;定义实型数组 c,有 20 个元素
char ch[20];                 //定义字符数组 ch,有 20 个元素
```

对于数组定义，应注意以下几点：

(1) 数组的类型实际上是指数组元素的取值类型。对于同一个数组，其所有元素的数据类型都是相同的。

(2) 数组名不能与其他变量名相同。例如：

```
int main()
{
    int a;
    float a[10];             //错误,数组名和变量名相同
    …
    return 0;
}
```

(3) 方括号中常量表达式表示的是数组元素的个数，如 a[5]表示数组 a 有 5 个元素。但是其下标从 0 开始计算，因此这 5 个元素分别为 a[0]、a[1]、a[2]、a[3]、a[4]。

(4) 不能在方括号中用变量来表示数组元素的个数，但可以用符号常数或常量表达式。例如：

```
#define D 5
int main()
{
    int a[3+5],b[4+D];       //合法的定义
    …
    return 0;
}
```

而下述定义方式是错误的：

```
int main()
{
    int n=10;
```

```
    int a[n];                    //不合法的定义,因为 n 为变量
    …
    return 0;
}
```

2. 一维数组元素的存储

每个数组元素都占用内存中的一个存储单元,每个元素都是一个变量,可以像以前讲过的普通变量一样使用,只不过数组元素是通过数组名和方括号中的下标来确定的。系统在内存中为数组元素分配连续的存储单元。

例如,定义语句 int a[15],声明了以下几个问题:

(1) 数组名为 a。

(2) 数组元素的数据类型为 int。

(3) 数组元素的下标值从 0 开始。数组元素的个数为 15,分别是 a[0]、a[1]、a[2]、…、a[13]、a[14]

(4) 数组名 a 是数组存储区的首地址,即存放数组第一个元素的地址。a 等价于 &a[0],因此数组名是一个地址常量。不能对数组名进行赋值或运算。在这个例子中,数据元素的存储形式如图 5.1 所示。

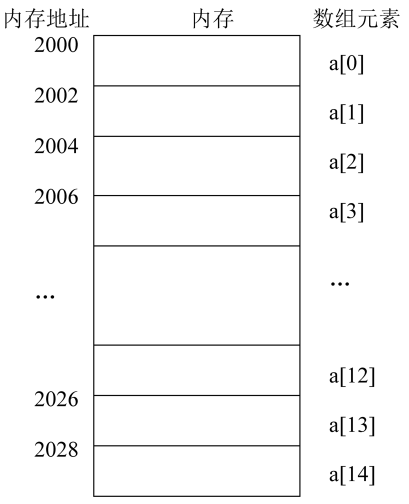

图 5.1 一维数组的存储形式

3. 一维数组元素的引用

对数组元素的引用与对变量的引用类似,但与变量引用不同的是,只能对数组元素进行引用,而不能一次引用整个数组。一维数组元素的引用格式如下:

```
数组名[下标]
```

其中:

(1) 下标可以是整型常量或整型常量表达式,如 a[3]、a[3+2]。

(2) 下标还可以是整型变量或整型变量表达式,如 a[i]、a[i+j]、a[i++]。

(3) 如果下标是表达式,首先要计算表达式,计算的最终结果为下标值。

(4) 下标值从 0 开始,而不是从 1 开始。

(5) 数组的引用下标不能越限,如用 int a[15]定义的数组,引用时的下标不能超过或等于 15,即 a[15]=10 是错误的。

定义数组时使用的“数组名[常量表达式]”和引用数组元素时使用的“数组名[下标]”形式相同,但含义不同。在数组定义的方括号中给出的是数组的长度,而在数组元素中的下标是该元素在数组中的位置标识,下标取值范围为 0 到数组长度值减 1。前者只能是

常量，后者可以是常量、变量或表达式。

【例 5-1】 从键盘输入 10 个整数，求其中的最大数并输出。

【问题分析】

(1) 定义一个一维数组，数组长度为 10，用于存储从键盘输入的 10 个整数。

(2) 求一组数的最大数：首先假定第一个数最大，把该数放到变量 max 中，然后依次处理后面 9 个数，如果当前正在处理的这个数比 max 还大，则更改 max 的值为这个数。将数组与 for 循环相结合，可以轻松完成此任务。

解决该问题的算法流程图如图 5.2 所示。

【程序代码】

```
#include <stdio.h>
int main()
{
    int a[10];                          //定义整型数组 a
    int i,max;                          //定义循环控制变量 i 和存放最大数的变量 max
    printf("Please enter ten integers:\n");     //输出屏幕提示语
    for(i=0;i<=9;i++)                   //循环 10 次
        scanf("%d",&a[i]);              //从键盘接收数据,并存放到数组元素 a[i]中
    for(i=0;i<=9;i++)                   //循环 10 次
        printf("%d  ",a[i]);            //输出数组元素 a[i]的值
    max=a[0];                           //给 max 变量赋值,假定第一个数最大
    for(i=1;i<=9;i++)                   //循环 9 次
        if(max<a[i]) /* 比较 max 与数组中当前正在处理的数组元素的大小,将较大者赋给
                        max */
            max=a[i];
    printf("\nThe max is %d\n",max); //输出求得的最大数
    return 0;
}
```

【运行结果】

程序运行结果如图 5.3 所示。

【代码解析】

(1) main()函数中的 int a[10]是数组的定义语句，定义一个长度为 10 的整型数组，用于存储从键盘输入的 10 个整数。

(2) main()函数中的第一个 for 循环，实现从键盘接收 10 个整数，并存放到数组 a 中。

(3) main()函数中的第二个 for 循环，实现把 10 个整数输出到屏幕。

(4) main()函数中的第三个 for 循环，实现求 10 个整数中的最大数。在此段代码中，变量 max 用来存放当前已比较过的各数中的最大数。开始时设 max 的值为 a[0]，然后将 max 与 a[1]比较，如果 a[1]大于 max，则 max 重新取值为 a[1]，下一次以 max 的新值与 a[2]比较，如果 a[2]大于 max，则 max 重新取值为 a[2]，此时 max 的值是 a[0]、a[1]、a[2]中的最大数。以此类推，经过 9 轮循环的比较，max 最后的值就是 10 个数中的最大数。

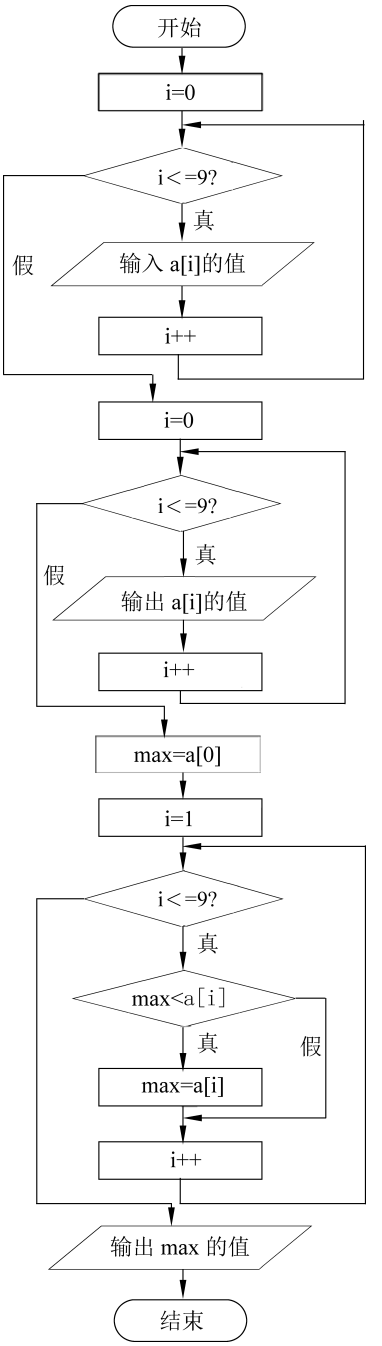
开始
i=0
i<=9?
真
假
输入 a[i]的值
i++
i=0
i<=9?
真
假
输出 a[i]的值
i++
max=a[0]
i=1
i<=9?
真
max<a[i]
真
假
max=a[i]
i++
输出 max 的值
结束

图 5.2　例 5-1 的流程图

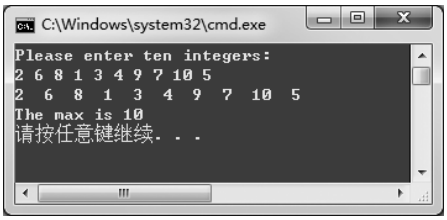

图 5.3　例 5-1 程序运行结果

(5) 最后一条输出语句输出求得的最大数。

说明：此示例只是为了演示一维数组的定义与引用。在实际问题中，如果只是要求找出一组数中的最大或者最小数，那么这组数据可以不保存，采用边读边求最大或者最小数的方式，程序更简洁、高效，实现代码参见例 4-7。

【例 5-2】 有等差数列 $a_n=5\times n+1$，要求分别按正序和逆序输出该数列的前 10 项到屏幕。n 为 0～9 的整数。

【问题分析】

(1) 计算数列的第 n 项的值：通过示例中给出的数列通项公式计算即可。

(2) 由于示例要求逆序输出数列的前 10 项，故需要一个长度为 10 的整型数组来存储数列的前 10 项的值。

(3) 数组与 for 循环相结合，完成数列前 10 项的计算、存储、正序和逆序输出。

【程序代码】

```
#include <stdio.h>
int main()
{
    int a[10];                          //定义整型数组 a
    int n;                              //定义循环控制变量 n
    for(n=0;n<=9;n++)                   //循环 10 次,计算数列的前 10 项,并按正序输出
    {
        a[n]=5*n+1;                     //计算数列的第 n 项的值,并赋给第 n 个数组元素
        printf("a[%d]=%d ",n,a[n]);     //输出数列的第 n 项的值
    }
    printf("\n");                       //换行
    for(n=9;n>=0;n--)                   //循环 10 次,逆序输出数列的前 10 项
    {
        printf("a[%d]=%d ",n,a[n]);
    }
    printf("\n");
    return 0;
}
```

【运行结果】

程序运行结果如图 5.4 所示。

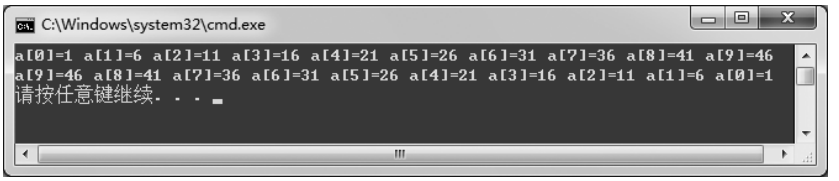

图 5.4　例 5-2 程序运行结果

【代码解析】

(1) mian 函数中的语句 int a[10],定义一个长度为 10 的整型数组 a,用于存储数列的前 10 项。

(2) main 函数中的第一个 10 次的 for 循环,循环控制变量从 0 开始依次递增 1,实现求数列的前 10 项,数列的第 n 项存储到数组元素 a[n]中,并正序输出。

(3) main 函数中的第二个 10 次的 for 循环,循环控制变量从 9 开始依次递减 1,实现逆序输出数列的前 10 项。

5.1.2　一维数组的初始化

与一般变量的初始化一样,数组的初始化就是在定义数组的同时,给其数组元素赋初值。数组初始化是在编译阶段进行的,这样将减少运行时间,提高效率。

一维数组初始化赋值的一般形式为:

```
类型声明符 数组名[常量表达式]={数值 1,数值 2,…,数值 n};
```

其中赋值号右边大括号中的各数据值即为各数组元素的初值,各值之间用逗号间隔。例如:

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

相当于

```
a[0]=0;a[1]=1;a[2]=2;
```

C 语言对数组的初始化有以下几点规定:

(1) 可以只给部分数组元素赋初值。当大括号中值的个数少于数组元素个数时,只为前面部分数组元素赋值。例如:

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

相当于只为 a[0]、a[1]、a[2]、a[3]、a[4]赋初值,而后 5 个元素自动赋为 0 值。

(2) 只能给数组元素逐个赋值,不能给数组整体赋值。例如,给 10 个元素全部赋值为 1,只能写为:

```
int a[10]={1,1,1,1,1,1,1,1,1,1};
```

而不能写为：

```
int a[10]=1;
```

(3) 如给全部元素赋值，则在数组声明中，可以不给出数组元素的个数。例如：

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

可写为：

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

(4) 大括号中数值的个数多于数组元素的个数是语法错误。

5.1.3 一维数组应用举例

【例 5-3】 参照例 4-16 中提出的兔子问题，应用数组计算 2 年后有多少对兔子？

【问题分析】

根据题意，以 f[n]表示 n 个月以后兔子的总对数，其规律为 f[n]＝f[n－2]＋f[n－1]，如此构成的数列如下：

f[1]＝1，f[2]＝1，f[3]＝2，f[4]＝3，f[5]＝5，…，f[n]＝f[n－2]＋f[n－1]，…

即斐波那契数列。此问题转化为求斐波那契数列第 24 项的值，其算法流程图如图 5.5 所示。

【程序代码】

```
#include <stdio.h>
int main()
{   int n, f[25];                          //定义整型变量 n,整型数组 f
    f[1]=f[2]=1;                           //为 f[1]和 f[2]赋初值 1
    for(n=3;n<=24;n++)                     //循环 22 次
        f[n]=f[n-1]+f[n-2];                //计算 f[n]的值
    for(n=1;n<=24;n++)                     //输出数列中所有项的值
    {   if((n-1)%4==0)                     //每输出 4 个数后换行
            printf("\n");
        printf("%10d",f[n]);               //输出数列中第 n 项的值
    }
    printf("\n");
    return 0;
}
```

【运行结果】

程序运行结果截图如图 5.6 所示。

说明：很多数列的问题都可以类似于上述的斐波纳契数列的计算方法，用数组来进行存储和计算。

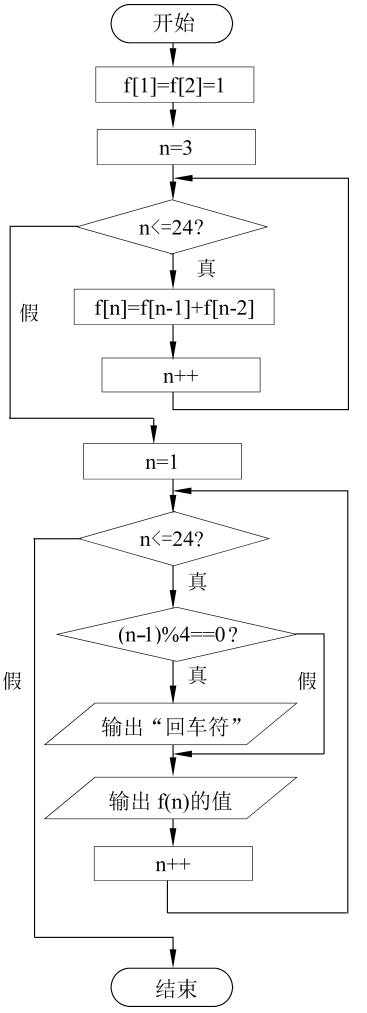

图 5.5　例 5-3 的流程图

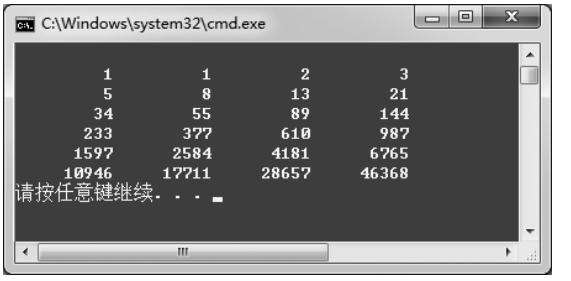

图 5.6　例 5-3 程序运行结果

【例 5-4】 给定 n 个任意数，按由小到大对其排序，并输出排序结果。

【问题分析】

这个问题是一组数的排序问题，排序方法有多种，这里采用冒泡排序法。

冒泡排序法的思路是，将相邻两个数比较，将小的数调到前头（或将大的数调到后面）。比如有 5 个数，分别是 7、6、10、4、2，依次将其放入数组 a 中。我们看一下冒泡排序法的处理过程。

(1) 第一趟（如图 5.7 所示），经过 4 次比较。

第 1 次：将第 1 个数 a[0]和第 2 个数 a[1]进行比较，将较大的数调到下面。也就是说，若后面的数小，就将两数交换，否则不交换。这里把 7 和 6 对调位置，结果如图 5.7(b)所示。

第 2 次：将第 2 个数 a[1]和第 3 个数 a[2]进行比较，将较大的调到下面。这里是 7 和 10 比较，这次比较不用对调这两个元素的位置，结果如图 5.7(c)所示。

第 3 次：将第 3 个数 a[2]和第 4 个数 a[3]进行比较，将较大的调到下面。这里是 10 和 4 比较后对调位置，结果如图 5.7(d)所示。

第 4 次：将第 4 个数 a[3]和第 5 个数 a[4]进行比较，将较大的调到下面。这里是 10 和 2 比较后对调位置，结果如图 5.7(e)所示。

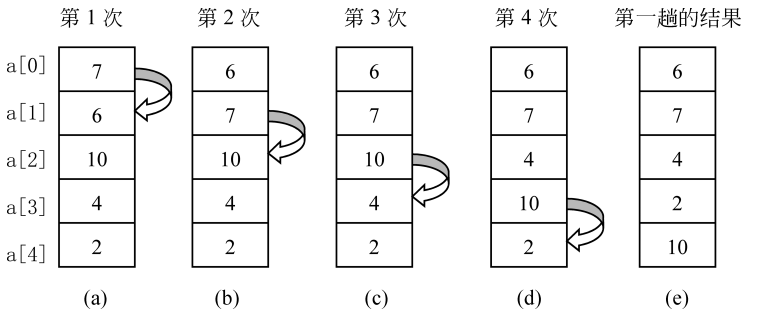

图 5.7 冒泡排序的第一趟过程

如此进行共 4 次，结果得到 6-7-4-2-10 的顺序，最大的数 10 成为最下面的一个数。最大的数“沉底”，最小的数 2 向上“浮起”一个位置（冒第一个泡）。

这 4 次处理过程都是类似的，都是“相邻两数比较，若后面的数小，则两数交换，否则不交换”；所不同的是“比较的两个数，它们的位置不同”，先是 a[0]和 a[1]比较，再是 a[1]和 a[2]比较，然后是 a[2]和 a[3]比较，最后是 a[3]和 a[4]比较。我们会发现一个规律，每次比较后，往下移了一位，所以可以用一个变量 i 控制，每次都是 a[i]和 a[i+1]比较，而每次比较完后 i+1，总共比较 4 次，这样就可以用一个循环来实现，即：

```
for(i=0;i<4;i++)
    if(a[i]>a[i+1])
    {
        temp=a[i]; a[i]=a[i+1]; a[i+1]=temp;
    }
```