第5章

数组和字符串

数组是变量的延伸，是同类型变量的集合。程序设计中使用数组可以有效地组织循环，从而使算法和编程大为简化。甚至有的问题，不用数组就难以解决。此外，字符串的存储和处理也是借助于字符型数组进行的。本章介绍有关数组及字符串的操作及应用。

5.1 数组的概念

1. 什么是数组

数组是同类型变量的集合。不同的变量可以用不同的标识符表示，而数组则用一个标识符和不同的下标来表示一批同类型的变量。例如，定义一个整型数组 a[10]，标识符 a 称为数组名，它包含 10 个变量，每个变量称为一个数组元素，第一个数组元素记为 a[0]，第二个数组元素记为 a[1]，…，第 10 个数组元素记为 a[9]。也就是说，数组中的各个元素都共用一个数组名，用方括号里的数字表示该元素在数组中的位置，称为下标。正由于数组元素不仅可以像简单变量那样存放一个值，还包含其在数组中位置的信息，因此人们也称数组元素为下标变量或矢量。

2. 数组的维数

维的概念类似于几何空间的概念，就像表示一维空间中的点要用一个坐标，表示二维空间的点要用两个坐标一样，一维数组的元素有一个下标，二维数组的元素有两个下标，三维数组有三个下标，依此类推……C 语言对数组的维数没有限制，但一维和二维数组使用频率最高。

【概念应用】

下面以一维数组和二维数组为例，介绍数组的逻辑结构和相关概念。

（1）一维数组用来描述一批有序的变量，例如，定义一个一维数组 a[10]，它描述顺序存储的 10 个变量，如图 5-1 所示。

（2）二维数组用来描述一个二维表，例如，定义一个二维数组 b[3][4]，它描述一个 3 行 4 列的二维表（或称矩阵），如图 5-2 所示。

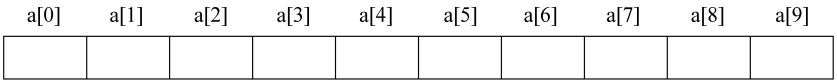


图 5-1　一维数组 a[10]

	第0列	第1列	第2列	第3列
第0行	a[0][0]	a[0][1]	a[0][2]	a[0][3]
第1行	a[1][0]	a[1][1]	a[1][2]	a[1][3]
第2行	a[2][0]	a[2][1]	a[2][2]	a[2][3]

图 5-2　3 行 4 列的二维表

其中，数组元素的第一个下标称为行下标，第二个下标称为列下标。

3. 数组的数据类型

数组的数据类型表示该数组中各个元素的类型，它可以是各种基本数据类型，如 int、float、double、char、long 等。

知识拓展

物以类聚，人以群分

C 语言规定，同一数组中的所有元素必须是相同数据类型的，通过不同的下标变量来引用每个数组元素。C 语言对于数组元素类型的要求与成语“物以类聚，人以群分”类似。该成语出自《战国策·齐策三》，用于比喻同类的东西常聚在一起，志同道合的人相聚成群，反之就分开。

5.2　数组的定义和机内存储

5.2.1　数组的定义

定义数组的一般形式如下：

```
[存储类型]  数据类型  name[expn][expn-1]…[exp1]
```

其中，存储类型是可选的，可以是 auto、static 或 extern；数据类型是必须有的；name $[exp_n][exp_{n-1}]\cdots[exp_1]$ 称为数组说明符，name 为数组名，exp_i 为第 i 维的维界表达式，规定该维可容纳的元素个数，必须是正整数常量。一个数组中元素的总和称为该数组的体积。

【概念应用】

下面具体介绍数组定义的具体应用和注意事项。

(1) 定义一维数组：数组说明符中只用一个维界表达式，并放在方括号中。例如：

```
double data[10];
```

定义了一个 double 型的含有 10 个元素的一维数组。

(2) 定义二维数组：数组说明符中要用两个维界表达式，并分别放在两个方括号中。例如：

```
float a[3][4];
```

定义了一个 float 型的 3 行 4 列共 12 个元素的二维数组。

(3) 定义 n 维数组：数组说明符中要用 n 个维界表达式，并分别放在 n 个“[]”中。例如：

```
int c[3][4][5];
```

定义了一个 int 型的含 60 个元素的三维数组。

(4) 数组的递归定义：C 语言对多维数组采用递归定义的方式，即用一维数组定义二维数组，用二维数组定义三维数组等。例如，二维数组 a[3][4]是一个含三个元素的一维数组，该数组的每个元素 a[0]、a[1] 和 a[2]又各是一个含 4 个元素的一维数组；三维数组 c[3][4][5]是一个含三个元素的一维数组，该数组的每个元素 c[0]、c[1]和 c[2]又各是一个 4 行 5 列的二维数组。根据递归定义，我们可以把三维数组 c[3][4][5]理解成三页，且每页都是一个 4 行 5 列的表格。

(5) 定义字符型数组：由于 C 语言没有字符串变量，字符串的存储和处理是借助于字符型数组进行的。因此，当需要对字符串进行处理时，就要定义字符型数组。通常用一维字符型数组存放一个字符串，用二维字符型数组存放若干个字符串。例如，定义

```
char str1[10];
char str2[2][11];
```

后，一维数组 str1 可存放一个字符串，该字符串最多不超过 9 个字符；二维数组 str2 可存放 2 个字符串，每个字符串的长度都不超过 10。其中，为了给字符串末尾的“\0”字符留出存储位置，在定义字符型数组时，数组的维界至少要比字符串的长度(即字符串中包含的字符个数)多 1。

(6) 定义数组应注意的几个问题。

① 定义数组时，数组说明符中的维界表达式只能是由常数或符号常数组成的整型表达式，且“[]”不能缺少。

② C 语言规定，数组的第一个元素总是从 0 下标开始。

③ 一个语句可以定义多个相同类型的数组或变量，相互之间用逗号隔开。例如，

```
int ch[20],v[4][12],a,x,y;
```

5.2.2 数组的存储

数组定义后，C 编译系统将为它分配一片连续的存储单元，依次存放它的各个元素。数组在内存中所占用的连续存储空间的首字节地址称为数组的首地址。该地址是 C 编译系统在编译时确定的，数组名就代表这个首地址，是一个常数。

【概念应用】

(1) 定义一维数组

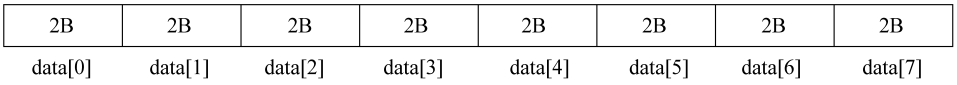
```
short int data[8];
```

一维数组 data 按数组元素的顺序依次存放，参见图 5-3（a）。

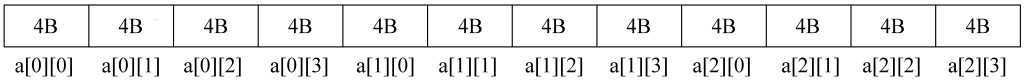
（2）定义二维数组

```
float array[3][4];
```

二维数组 array 存放时，先存放第 0 行的各个元素，再存放第 1 行的各个元素……这种存放方式叫作“按行存放”，即二维数组按行方式线性存放，如图 5-3(b)所示。



(a) 一维数组data[8]



(b) 二维数组array[3][4]

图 5-3　数组的存储

5.3　数组的初始化

和变量一样，数组也可以在定义的同时，给各个数组元素赋以初值。

5.3.1　一维数组的初始化

一维数组的初始化，要把全部初值顺序放在“＝”右边的“{}”中，各初值之间用“,”隔开。“{}”连同其中的初值作为一个整体，称为初值表。

【概念应用】

```
int data[8]={50,60,70,80,90,100,110,120};
```

编译时，编译系统会自动地从第一个元素开始，将“{}”中的常数顺序存放在各个数组元素的存储单元中。本例中，data[0]初值为 50，…，data[7]初值为 120。

5.3.2　二维数组的初始化

二维数组的初始化，要把全部初值放在“＝”右边的“{}”中，各行的初值又分别放在一对内嵌的“{}”中。

【概念应用】

经下面的定义

```
float array[3][4]={{1.0,2.0,3.0,4.0}, {5.0,6.0,7.0,8.0}, {9.0,10.0,11.0,12.0}};
```

后，初始化的结果如下：array [0][0]＝1.0，array [0][1]＝2.0，…；array [1][0]＝5.0，array [1][1]＝6.0，…，array [2][3]＝12.0。

如果初值的顺序与数组元素的顺序一致，则代表每一行的内嵌的“{}”也可以省略，上面的定义可写成

```
float array[3][4]={1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0,12.0};
```

5.3.3 字符型数组的初始化

字符型数组既可以用字符常数，也可以用字符串常数对字符型数组初始化。

1. 用字符常数初始化

当用字符常数初始化时，应将字符常数依次排列，相互之间用“,”隔开，并用“{}”括住，特别要注意各个字符串末尾的'\0'字符不能缺少。

【概念应用】

```
char str[12]={'T','h','e',' ','s','t','r','i','n','g','.','\0'};
char Language[5][8]={{'B','A','S','I','C','\0'},
                     {'F','O','R','T','R','A','N','\0'},
                     {'P','A','S','C','A','L','\0'},
                     {'C','\0'},
                     {'C', 'O','B','O','L','\0'}};
```

2. 用字符串常量初始化

比较简便的方法还是直接用字符串常数进行初始化。这时，应直接将字符串常量放在“=”右边的“{}”中(“{}”也可以省略)，而且不必添加字符串末尾的'\0'字符。

【概念应用】

```
char str[12]={"The string."};或 char str[12]="The string.";
char Language[5][8]={"BASIC","FORTRAN","PASCAL","C","COBOL"};
```

5.3.4 有关数组初始化的几点说明

(1) C 语言规定：允许对全局数组、static 型局部数组及 main()中的 auto 型数组进行初始化，而 extern 型和其他非 main()函数中的 auto 型数组不能进行初始化。

(2) 用于初始化的数据只能是常数表达式(含常数和符号常数)。

(3) 允许只对数组前面若干个连续的元素赋初值，这时，后面的元素将自动赋 0 值。例如：

```
int x[6]={1,2,3,4};
```

由于数组 x 有 6 个元素，所给初值只有 4 个，分别赋给数组的前 4 个元素，其余的元素将取 0 值，即 x[0]=1，x[1]=2，x[2]=3，x[3]=4，x[4]=0，x[5]=0。又如：

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

由于用内嵌的“{}”限制了各行的初值，各行的元素将被顺序赋值，余下的元素赋 0 值，它等价于

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

(4) 若所给初值的个数多于定义的数组元素个数，编译系统将报错。例如，若在

main()中定义：

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

则编译时将显示

```
error C2078: 初始值设定项太多
```

的错误信息。

(5) 字符型数组初始化时，若初值字符个数多于定义的数组长度时，虽然 Visual C++ 2010 系统编译时不报错，但给出警告(warning C4045：数组界限溢出)，也不能显示正确的值；如果初值字符的个数少于数组长度时，在数组末尾自动填充'\0'字符。例如，定义

```
char str1[10]="morning";
```

后，存储内容如图 5-4 所示。

m	o	r	n	i	n	g	\0	\0	\0

图 5-4　str1[10]的存储内容

又如，定义

```
char str2[2][12]={ "afternoon","he waked up"};
```

后，存储内容如图 5-5 所示。

a	f	t	e	r	n	o	o	n	\0	\0	\0	h	e		w	a	k	e	d		u	p	\0

图 5-5　str2 的存储内容

5.3.5　初始化定义数组

初始化定义数组主要是解决隐含尺寸数组定义问题。隐含尺寸数组是指在定义数组时并没有指定数组的大小，而是用初始化中初值的个数隐含规定数组的大小。

1. 一维隐含尺寸数组的定义

定义一维隐含尺寸数组时，数组说明符中的维界表达式被省略，只保留数组名及其后的“[]”，数组的维界则由初值表中初值的个数隐含规定。

【概念应用】

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

中，初值的个数为 8，C 编译系统就自动为数组 a 安排能存放 8 个整型数据的连续存储空间，因此，它与

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

具有相同的效果。

需要强调的是，C 语言规定任何数组必须有确定的大小，编译系统才能为它安排足够大小的连续存储空间。隐含尺寸数组并不是定义了一个不确定大小的数组，而是通过初值的个数隐含地规定了数组的大小。一维数组如此，二维数组及多维数组也是如此。

2. 二维隐含尺寸数组的定义

定义二维隐含尺寸数组时，省略数组说明符中第一个维界表达式，只指明第二个维界表达式，而将各行的初值分别放在内嵌的花括号中。

【概念应用】

(1) 在语句

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

中，只规定了第二维的维界为 4，而第一维的维界则由初值表中内嵌的“{}”的数量来确定。它等价于

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

或

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

(2) 如果在定义二维隐含尺寸数组时，初值表中未使用内嵌的“{}”，例如：

```
int b[][4]={1,2,3,4,5,6,7};
```

这时，编译系统就根据给定的初值个数和第二维的维界来确定第一维的维界：若初值的个数能被第二维的维界整除，则所除得的商就是第一维的维界；否则，所除得的商的整数部分加上 1 就是第一维的维界。根据这一规则，上面定义中第一维的维界为 7/4＋1＝2，因此，它与

```
int b[2][4]={1,2,3,4,5,6,7};
```

等价。

(3) 和二维隐含尺寸数组一样，在定义任何多维的隐含尺寸数组时，都只能省略第一维的维界，其他各维的维界一律不能省略。

3. 字符型隐含尺寸数组的定义

定义隐含尺寸的字符型数组，可以免除乏味的统计字符个数的工作。

【概念应用】

(1) 在语句

```
char str[]="The string.";
```

或

```
char str[]={'T','h','e',' ','s','t','r','i','n','g','.','\0'};
```

编译系统会根据初值给出的字符个数自动为该数组分配 12 字节的存储空间。

(2) 在语句

```
char language[][8]={"BASIC","FORTRAN","PASCAL","C","COBOL"};
```

中，字符串常数的个数就是第一维的维界。

关于字符型隐含尺寸数组，应注意两点。

(1) 在用字符常量定义隐含尺寸的字符型数组时,初值表中字符的个数直接影响数组的维界,如果遗漏了末尾的'\0'字符,就会使数组减少 1B 的存储空间;而用字符串常量定义隐含尺寸的字符型数组时,编译系统会自动为数组增加 1B 的存储空间。例如:

```
char a[]={'V','i','s','u','a','l','','C','+',+'},b[]="Visual C++";
```

虽然这两个字符串的长度都是 10,但数组 a 占用的内存空间是 10B,而数组 b 占用的内存空间是 11B。

(2) 在数组维界显式确定的情况下,数组所占存储空间的大小是由数组定义的大小决定的,而不是由初始化字符串的长度决定的。例如:

```
char st[20]="Visual C++";
```

数组 st 所占的存储单元的个数是 20 而不是 11。

5.4 数组的基本操作

数组的基本操作包括数组元素的引用、数组的赋值、数组的输入输出等。

5.4.1 数组元素的引用

1. 一维数组元素的引用

用"数组名[下标表达式]"的形式来引用一维数组元素,其中,下标表达式必须放在方括号内,且只能取整型值。下标的下限是 0,而上限不能超过该数组定义时的维界值减 1。由于编译系统不检查下标是否越界,因此使用了越界的下标后,可能破坏其他数据甚至程序代码。

2. 二维数组元素的引用

用"数组名[下标表达式 1][下标表达式 2]"的形式来引用二维数组元素,其中,"下标表达式 1"表示行下标,"下标表达式 2"表示列下标,二者必须分别放在"[]"内。关于下标表达式类型的限制及下标越界的处理与一维数组相同。

3. 字符型数组元素的引用

可以用与上面类似的形式来引用一维或二维字符型数组的元素,即字符串中的一个字符。

5.4.2 数组的赋值

数组的赋值只能对数组元素逐个赋值,不能直接对数组名赋值。

【概念应用 1】

```
int i,a[5];
a[0]=100,a[1]=120,a[2]=200,a[3]=250,a[4]=500;
```

上面的赋值语句不能写成：

```
a={100,120,200,250,500};          //这是错误的数组赋值方式
```

【概念应用 2】

如果所赋的值有某种规律，可以借助于循环结构来简化程序的编制。

```
int b[3][4],i,j;
    for (i=0;i<3;i++)
        for (j=0;j<4;j++)
            b[i][j]=i+j;
```

注意，本例 for 语句的循环条件测试表达式不能写成 i＜＝3 及 j＜＝4，这样会造成下标越界。

【概念应用 3】

对字符型数组的赋值，只能对每个元素用字符常量赋值。例如要把字符串"C++"赋给字符型数组 st，可用如下程序段完成：

```
char st[4];
st[0]='C';st[1]='+';st[2]='+',st[3]='\0';
```

不能将上面的赋值语句写成下列任何一种形式：

```
st[]="C++";
st[4]="C++";
st="C++";
```

注意：对数组赋值与数组初始化是两种不同的操作。用上面的方法将一个很长的字符串赋给一个字符型数组确实是太烦琐了，在实际使用中，经常用下面将要介绍的字符串复制函数 strcpy()来把字符串常量复制到字符型数组中，达到赋值的效果。

5.4.3 数组的输入和输出

1. 一维数组的输入和输

对一维数组，一般用单循环实现对数组各个元素逐个输入或逐个输出。

【概念应用】

```
float x[10];
int i;
for(i=0;i<10;i++)
    scanf("%f",&x[i]);
for(i=0;i<10;i++)
    printf("%f   ",x[i]);
```

其中，作为 scanf()函数的输入项，x[i]前面一定要加上取地址运算符“&”，表示元素 x[i]的地址。程序运行时，从键盘顺序输入各个元素的值，每两个输入数据之间可以用空格键、回车键或制表符来分隔，最后以回车键结束。

2. 二维数组的输入和输出

对二维数组，通常用二重循环实现对数组各个元素逐个输入或输出。输入时，既可以

按行优先的顺序，也可以按列优先的顺序进行。

【概念应用 1】

按行优先顺序输入，例如：

```
int a[3][4],i,j;
for (i=0;i<3;i++)
    for (j=0;j<4;j++)
        scanf("%d",&a[i][j]);
```

外层循环控制行数的变化，内层循环控制列数的变化。行数每变化一次，列数要变化 4 次。因此，程序运行时，从键盘先输入第 1 行各元素的值，然后输入第 2 行各元素的值……直到最后一行的数据输入完毕。每两个输入数据之间可以用空格符、回车符或制表符分隔，最后以回车符结束。

【概念应用 2】

按列优先顺序输入，例如：

```
int a[3][4],i,j;
for(j=0;j<4;j++)
    for(i=0;i<3;i++)
        scanf("%d",&a[i][j]);
```

外层循环控制列数的变化，内层循环控制行数的变化。列数每变化一次，行数要变化 3 次。因此，程序运行时，从键盘先输入第一列各元素的值，然后输入第二列各元素的值，……直到最后一列的数据输入完毕。每两个输入数据之间可以用空格符、回车符或制表符来分隔，最后以回车符结束。

【概念应用 3】

输出二维数组时，一般要按行的顺序使输出的结果能呈现二维表格的形式，即位于数组同一行的各元素在屏幕也显示在同一行上。例如：

```
#include "stdio.h"
main()
{   int i,j;
    double b[3][4]={1.2,2.2,3.4,4.4,5.6,6.6,7.6,8.6,9.8,10.8,11.8,12.8};
    for (i=0;i<3;i++)
    {   for (j=0;j<4;j++)
            printf("%6.2f",b[i][j];          //内循环语句
        printf("\n");                        //外循环语句
    }
}
```

程序中的第一个 printf()位于内层循环体内，而第二个 printf()位于外层循环体内，当内层循环变量 j 从 0 变化到 3 时，所有的 b[i][j]都显示在同一行上。当退出内层循环时，首先执行 printf("\n")，结果换入下一行，然后进入外层循环的下一轮循环，输出下一行的各个元素值。程序运行结果如下：

```
1.20    2.20    3.40    4.40
5.60    6.60    7.60    8.60
9.80    10.80   11.80   12.80
```