第5章　数　　组

实验目的

- 掌握一维数组和二维数组的定义、赋值和输入输出的方法。
- 掌握应用一维数组的基本算法，实现对一维数组的置数、逆置、查找、排序等操作。
- 掌握应用二维数组的基本算法，了解其与矩阵的关系，实现对二维数组进行置数，对每行每列或指定行列的元素进行处理。
- 掌握字符数组和字符串函数的使用。

5.1　数组基本知识提要

5.1.1　一维数组

1. 一维数组的定义

一维数组是指每个元素只有一个下标的数组。在C语言中，定义一维数组的语句形式如下：

```
类型名　数组名[整型常量表达式];
```

例如：

```
int a[10];
```

说明：

(1) a为一维数组的数组名。

(2) a数组含有10个元素，它们分别是a[0]，a[1]，a[2]，…，a[9]。

(3) a数组中的每个元素都是整型，即每个元素中只能存放整型数据。

(4) 每个元素只有一个下标，C语言规定每个数组第一个元素的下标为0，这里a数组中最后一个元素的下标为9，即该数组下标的上限为9。

(5) C编译程序为a数组在内存中开辟10个连续的存储单元，如图5-1所示。

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

图5-1　数组a在内存中开辟的存储单元示意图

(6) 在一个数组定义语句中，可以有多个数组说明符，它们之间用逗号隔开。例如：

```
float x[10],y[15],z[20];
```

定义了名为x、y、z的3个实型数组。其中，x数组中含有10个元素，x数组的下标上限为

9;y 数组中含有 15 个元素,y 数组的下标上限为 14;z 数组中含有 20 个元素,z 数组的下标上限为 19。

(7) 数组说明符和普通变量名也可同时出现在一个类型定义语句中。例如:

```
char c1,c2,str[80];
```

注意:数组说明符的一对中括号中只能是整型常量或整型常量表达式,不能是变量。上述定义语句还可以写成

```
char c1,c2,str[50+30];
```

2. 一维数组元素的引用

一维数组元素的引用形式为

```
数组名[下标表达式]
```

例如,若有以下定义语句:

```
int a[10];
```

则 a[0]、a[i]、a[i+k]都是对 a 数组中的元素的合法引用形式,其中 0、i、i+k 称为下标表达式。由于定义了数组 a 有 10 个元素,因此各下标表达式的值必须是整数,大于或等于 0,并且小于 10。

注意:

(1) 一个数组元素实质上就是一个变量名,代表内存中的一个存储单元。一个数组占用一片连续的存储单元。

(2) C 语言中,数组名中存放的是一个地址常量,它代表整个数组的首地址。

(3) 只能逐个引用各个数组元素,不能一次引用整个数组。例如上述定义的 a 数组,不能用数组名 a 代表 a[0]到 a[9]这 10 个元素。

(4) 引用数组元素时,数组的下标可以是整型常量或整型表达式,下标的下限为 0。

(5) 在 C 语言程序运行过程中,编译系统并不检查数组元素的下标是否越界,数组的上、下限均有可能越界,从而可能破坏其他存储单元中的数据或程序代码。因此,编写程序时需保证数组下标不能越界。

3. 一维数组初始化

定义数组时,系统为该数组在内存中开辟一片连续的存储单元,但这些存储单元中并没有确定的值。可以通过以下几种方式对数组元素赋初值:

(1) 在定义数组时对数组元素赋初值。例如:

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

所有元素的初值放在赋值号后的一对大括号中,数值的类型必须与数组定义中说明的类型一致,各元素的初值之间用逗号隔开。系统将按这些数值的排列顺序,从 a[0]开始依次给个元素赋初值。上述语句就是将 a[0]赋初值 0,将 a[1]赋初值 1……将 a[9]赋

初值 9。

在指定初值时，第一个值必定赋给下标为 0 的元素，然后依次赋值，不可能跳过前面的元素给后面的元素赋初值。当所赋初值少于数组的元素个数时，将自动给后面的元素赋初值 0；当所赋初值个数多于数组的元素个数时，编译时，系统将给出出错信息。例如：

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

相当于

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

(2) 对于字符型数组，也同样对未赋初值的元素赋初值 0，即'\0'(其 ASCII 码值为 0)。例如：

```
char c[5]={ '*'};
```

相当于

```
char c[5]={ '*','\0','\0','\0','\0' };
```

(3) 可以通过赋初值来定义数组的大小。例如：

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

赋值号后面的一对大括号内有 10 个数值，即隐含定义了 a 数组含有 10 个元素。

4. 一维数组应用举例

【例 5-1】 编写程序，定义一个含有 30 个元素的 int 类型数组，依次给数组元素赋值为 1,3,5,…，然后分别以每行 10 个数输出，先顺序输出，再逆序输出。

本例展示了如何利用循环控制变量顺序或逆序逐个引用数组元素，以及在连续输出数组元素的过程中如何利用循环变量来进行换行控制。

程序代码：

```
#include <stdio.h>
#define M 30
main()
{  int s[M],i,k=1;
   for(i=0; i<M; i++)
   {  s[i]=k; k+=2;   }                  /*给数组元数依次赋值为 1,3,5,…*/
   printf("\n Sequence Output:\n");
   for(i=0; i<M; i++)                    /*从前往后顺序输出各元素值*/
   {  printf("%4d",s[i]);
      if((i+1)%10==0) printf("\n");      /*利用 i 控制换行符的输出*/
   }
   printf("\n Invert Output:\n");
   for(i=M-1; i>=0; i--)                 /*从后往前逆序输出各元素值*/
```

```
        printf("%3d%c",s[i],(i%10==0?'\n':' '));
    /*利用条件表达式来决定输出换行符还是输出空格*/
}
```

运行该程序,输出结果如下:

```
Sequence Output:
1   3   5   7   9   11  13  15  17  19
21  23  25  27  29  31  33  35  37  39
41  43  45  47  49  51  53  55  57  59
Invert Output:
59  57  55  53  51  49  47  45  43  41
39  37  35  33  31  29  27  25  23  21
19  17  15  13  11  9   7   5   3   1
```

【例 5-2】 已知某班 50 个学生的期末考试成绩分别为

43 65 51 27 79 11 56 61 82 9 25 36 7 49 55 63 74
81 49 37 40 49 16 75 87 91 33 24 58 78 65 56 76 67
45 54 36 63 12 21 73 49 51 19 39 49 68 93 85 59

编写程序,分别统计出成绩为 0～9,10～19,20～29,…,90～99,100 的学生人数。

程序代码:

```
#include <stdio.h>
#define MAXVAL 50
#define COUNTER 11
void main()
{
    float value[MAXVAL];
    int i,low,high;
    int group[COUNTER]={0,0,0,0,0,0,0,0,0,0,0};
    for(i=0; i<MAXVAL; i++)
    {  scanf("%f",&value[i]);              /*输入成绩*/
       ++group[(int)(value[i])/10];        /*对成绩分段处理*/
    }
    printf("\n");
    printf("GROUP        RANGE        FREQUENCY\n\n");
    for(i=0; i<COUNTER; i++)
    {
        low=i*10;
        if(i==10)
            high=100;
        else
            high=low+9;
        printf("%2d\t%3d\tto%3d\t%d\n",i+1,low,high,group[i]);
    }
}
```

程序的运行结果：

```
GROUP    RANGE                    FREQUENCY
1        0        to      9          2
2        10       to      19         4
3        20       to      29         4
4        30       to      39         5
5        40       to      49         8
6        50       to      59         8
7        60       to      69         7
8        70       to      79         6
9        80       to      89         4
10       90       to      99         2
11       100      to      100        0
```

5.1.2 二维数组

1. 二维数组的定义

当数组中每个元素带有两个下标时，称为二维数组。在逻辑上，也可以把二维数组看成是一个具有行和列的表格或矩阵。

在C语言中，二维数组中元素的排列顺序是按行存放。即在内存中先按顺序存放第一行的元素，再存放第二行的元素……因此，二维数组元素的存放方式与一维数组类似，总是占用一块连续的存储单元。

二维数组的定义语句形式为

```
类型名  数组名[常量表达式1][常量表达式2];
```

例如：

```
int a[3][4];
```

int是类型名，a[3][4]是二维数组说明符。从此定义语句可知：

(1) a是一个具有3行4列的二维数组a，注意不能写成a[3,4]。

(2) a数组中每个元素都是整型。

(3) a数组中有3×4=12个元素，a数组是一个具有如下形式的3行4列的矩阵或表格：

	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]

每个元素有两个下标：第一个中括号中的下标表示行号，称为行下标；第二个中括号

中的下标表示列号，称为列下标。a 数组的行下标下限为 0，上限为 2；列下标下限为 0，上限为 3。

a 数组中的元素在内存中占一片连续的存储单元，其排列顺序为按行存放，即先存放第 0 行的元素，再存放第 1 行的元素，最后存放第 2 行的元素，如图 5-2 所示。

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

图 5-2　二维数组 a 的元素在内存中的排列顺序示意图

在 C 语言中，可以把一个二维数组看成一个一维数组，只是该一维数组中的每个数组元素又是包含若干个元素的一维数组。例如，上述 a 数组可以看成是由 a[0]、a[1]、a[2]这 3 个元素组成的一维数组，其中每个元素又是由 4 个整型元素组成的一维数组。

2. 二维数组元素的引用

引用二维数组元素必须给出两个下标，引用形式为

```
数组名[下标表达式 1][下标表达式 2]
```

例如，若有以下定义语句：

```
double x[3][4];
```

则 x[0][1]、x[i][j]、x[i+k][j+k]都是合法的数组元素引用形式，只是每个下标表达式的值必须是整数，且不得超过数组定义中的上、下限。

注意：引用二维数组元素时，必须将两个下标分别放在两个中括号内。例如，引用上述 x 数组中的元素时，不可以写成 x[0,1]、x[i,j]、x[i+k,j+k]，这些都是不合法的。

3. 二维数组的初始化

二维数组的初始化赋值通常有以下几种形式：

(1) 所赋初始值个数与数组元素的个数相同。

可以在定义二维数组的同时给二维数组的各元素赋初值。例如：

```
int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

全部初值放在一对大括号中，每一行的初值又分别放在一对大括号中，大括号之间用逗号隔开。

(2) 每行所赋初值个数与数组元素的个数不同。

当某行一对大括号内的初值个数少于该行中元素的个数时。例如：

```
int a[3][4]={{1,2},{5},{9,10,11,12}};
```

系统自动将该行后面的元素赋初值 0。这里 a[0][2]、a[0][3]、a[1、1]、a[1][2]、a[1][3]这几个元素的初值均自动赋为 0。

(3) 所赋初值行数少于数组行数。

当代表着给每行赋初值的大括号数少于数组的行数时。例如：

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

系统将自动给后面各行的元素赋初值0。

(4) 赋初值时省略大括号对。

给二维数组赋初值时也可以不用大括号对。例如：

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

注意，此时所赋初值与情况(3)中所赋初值的结果完全不同。在这里是将a数组的第0行的4个元素依次赋值为1、2、4、5，即a[0][0]=1，a[0][1]=2，a[0][2]=4，a[0][3]=5，而数组中的其他元素的初值都为0。

(5) 通过赋初值定义二维数组的大小。

对于一维数组，可以在数组定义语句中省略中括号中的常量表达式，通过所赋初值的个数来确定数组的大小；对于二维数组，则只可以省略第一个中括号中的常量表达式，而不能省略第二个中括号中的常量表达式。例如：

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

上述语句中，a数组的第一维的中括号中的常量表达式省略了，在所赋初值中，含有3个大括号对，则第一维的大小由所赋初值的行数来决定。因此，它等同于

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

如果用以下形式赋初值：

```
int b[][3]={1,2,3,4,5,9};
```

第一维的大小按以下规则确定：

(1) 当初值的个数能被第二维的常量表达式的值整除时，所得的商就是第一维的大小。

(2) 当初值的个数不能被第二维的常量表达式的值整除时，则第一维的大小为商+1。

因此，按照此规则，上述b数组第一维的大小应该是2，也就是以上语句等同于

```
int b[2][3]={1,2,3,4,5,9};
```

4. 二维数组应用举例

【例5-3】 编程计算并输出9×9乘法表。

程序代码：

```
#include <stdio.h>
#define ROWS 9
#define COLUMNS 9
void main()
{  int row, column, product[ROWS][COLUMNS];
```

```
    int i,j;
    printf("      MULTIPLICATION  TABLE \n\n");
    printf("      ");
    for(j=1; j<=COLUMNS; j++)
    printf("%4d",j);                          /* 打印表头 */
    printf("  \n");
    printf("-------------------------------------------------\n");
    for(i=0; i<ROWS; i++)
    {   row=i+1;
        printf("    %2d |   ",row);
        for(j=1; j<=COLUMNS; j++)
        {   column=j;
            product[i][j]=row*column;
            printf("%4d",product[i][j]);
        }
        printf("\n");
    }
}
```

程序运行结果如下：

```
       MULTIPLICATION        TABLE
        1   2   3   4   5   6   7   8   9
-------------------------------------------------
1  |    1   2   3   4   5   6   7   8   9
2  |    2   4   6   8   10  12  14  16  18
3  |    3   6   9   12  15  18  21  24  27
4  |    4   8   12  16  20  24  28  32  36
5  |    5  10  15  20  25  30  35  40  45
6  |    6  12  18  24  30  36  42  48  54
7  |    7  14  21  28  35  42  49  56  63
8  |    8  16  24  32  40  46  56  64  72
9  |    9  18  27  36  45  54  63  72  81
```

5.1.3 字符串与字符数组

C 语言本身并没有设置一种类型来定义字符串变量，字符串的存储完全依赖于字符数组，但字符数组又不等同于字符串变量。

1. C 语言对字符串的约定

在 C 语言中，字符串是借助字符型一维数组来存放的，并规定以字符'\0'作为字符串结束标志。'\0'是一个转义字符，称为“空值”，它的 ASCII 码值为 0。'\0'作为标志也要占用存储空间，但不计入串的实际长度。

2. C语言对字符串常量的约定

虽然C语言中没有字符串数据类型,但允许使用字符串常量。字符串常量是由双引号括起来的一串字符,在表示字符串常量时,不需要人为在其末尾加入'\0',C编译程序将自动在字符串的末尾加入字符'\0'。

3. C语言中字符串常量给出的是地址值

每一个字符串常量都分别占用内存中一片连续的存储空间,这片连续的存储空间实际上就是字符型一维数组。这个数组没有名字,C编译系统以字符串常量的形式给出存放每一字符串的存储空间首地址,不同的字符串具有不同的起始地址。也就是说,在C语言中,字符串常量被隐含处理成一个以'\0'结尾的无名字符型一维数组。

4. 字符数组与字符串的区别

字符串是字符数组的一种具体应用。字符数组的每个元素中可存放一个字符,但它并不限定最后一个字符应该是什么。而在C语言中,因为有关字符串的大量操作都与串结束标志'\0'有关,因此,在字符数组中的有效字符后加上'\0',就可以把一维字符数组看作字符串变量。

注意:仅可以在字符数组内存放字符串,不能通过赋值语句将字符串常量或其他字符数组中的字符串直接赋值给字符串变量。

【例5-4】 编程实现以下功能:从终端输入一个字符串,将其复制到另一个字符数组中,并统计复制的字符个数。

程序代码:

```
#include <stdio.h>
#include <string.h>
void main()
{
    char string1[80],string2[80];
    int i;
    printf("Enter a string \n");
    scanf("%s",string1);
    for(i=0; string1[i] !='\0'; i++)
        string2[i]=string1[i];
    string2[i]='\0';
    printf("\n");
    printf("%s\n",string2);
    printf("Number of characters=%d\n",i);
}
```

运行程序,在键盘上输入Appleissweet后按回车键,输出

```
Number of characters=12
```

【例 5-5】 编程输出大小写 26 个字母及其对应的 ASCII 码值。

程序代码：

```
#include <stdio.h>
void main()
{
    char c;
    int i=0;
    printf("\n\n");
    for(c=65; c<=122; c++)
    {
        if(c>90&&c<97)
            continue;
        if(i%8==0) printf("\n");
          {  printf("|%2c--%4d",c,c);i++;  }
    }
    printf ("|\n");
}
```

运行程序，得到如下输出结果：

```
|A--  65  |B--  66  |C--  67  |D--  68  |E--  69  |F--  70  |G--  71
|H--  72  |I--  73  |J--  74  |K--  75  |L--  76  |M--  77  |N--  78
|O--  79  |P--  80  |Q--  81  |R--  82  |S--  83  |T--  84  |U--  85
|V--  86  |W--  87  |X--  88  |Y--  89  |Z--  90  |a--  97  |b--  98
|c--  99  |d--  100 |e--  101 |f--  102 |g--  103 |h--  104 |i--  105
|j--  106 |k--  107 |l--  108 |m--  109 |n--  110 |o--  111 |p--  112
|q--  113 |r--  114 |s--  115 |t--  116 |u--  117 |v--  118 |w--  119
|x--  120 |y--  121 |z--  122
```

5.2 排序与查找算法

5.2.1 排序算法

所谓排序，就是将一个任意顺序的数据元素序列重新排列成有序的序列。排序的算法有很多，对空间的要求及其时间效率也不尽相同。计算机程序设计中常用的排序方法有冒泡排序、选择排序、插入排序、快速排序、希尔排序、归并排序、基数排序等。其中，冒泡排序、选择排序、插入排序又被称作简单排序，它们对空间的要求不高，但是时间效率却不稳定；后面几种排序对空间的要求稍高一点，但时间效率能稳定在很高的水平。

1. 冒泡排序

冒泡排序算法基本思想是：对两个相邻的数比较大小，较大的数下沉或较小的数上