第5章　数　　组


5-0.mp4

前面已经学习了各种不同的数据类型，例如整型、实型、字符型、指针型，尽管这些类型在内存中所占的存储单元长度不同，但都只能表示一个大小或精度不同的数值，每一个值是不能分解的。而在实际应用中，常常要遇到要处理相同类型的成批相关数据的情况。程序设计语言为组织这类数据提供了一种有效的类型——数组。

本章将介绍数组的概念和基本应用。

5.1　一维数组

数组是由一定数目的同类元素顺序排列而成的数据集合。在计算机中，一个数组在内存占有一片连续的存储区域，C语言的数组名就是这块存储空间的首地址。数组的每个元素用下标变量标识，数组要求先定义后使用。

5.1.1　一维数组的定义、初始化

1. 一维数组的定义

定义一维数组的格式如下：

```
类型 标识符[表达式];
```

其中，“类型”指定了数组中每个元素的数据类型，“标识符”是用户自定义的数组名，代表数组的首地址；“[ ]”是数组类型符，用以说明“标识符”的类型；“表达式”为整型表达式，用于指定数组元素的个数，即数组的长度。一维数组只有一个下标表达式。

例如，如果存储100个学生的成绩，且成绩为实型数据，则数组定义如下：

```
float scores[100];
```

Dev-C++的数组下标从0开始。长度为 N 的数组，下标为0～$N-1$，注意“[ ]”中的整型表达式只表示数组元素的个数。数组scores中的元素与下标的对应关系如图5.1所示。

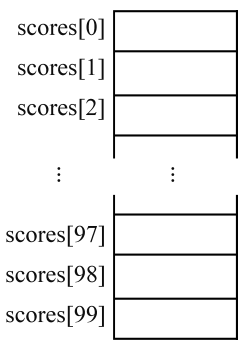

图5.1　一维数组scores中元素与下标的对应关系

在定义数组时需要注意，数据类型必须是已经定义的，下标表达式应当有确定的整数值，不能为实型表达式。

在Dev-C++中，允许定义数组时下标为整型变量，但这个整型变量必须已经赋值。例如：

```
int n;
scanf("%d",&n);
int a[n];
```

2. 一维数组的初始化

一维数组元素的初始化可以用下列方式表示。

(1) 在定义数组时,初始化数组元素。例如:

```
float c[5]={2,5,3.5,10.9,6};
```

是将数组 c 初始化后 $c[0]=2$,$c[1]=5$,$c[2]=3.5$,$c[3]=10.9$,$c[4]=6$。

(2) 可以只给数组的一部分元素赋值。例如:

```
int d[5]={0,1,3};
```

是将数组 d 有 5 个元素,经过初始化后,$d[0]=0$,$d[1]=1$,$d[2]=3$,后面没有赋值的两个数组元素 $d[3]$和 $d[4]$的值为 0。

(3) 在对数组元素赋初值时,可以不指定数组的长度。例如:

```
double scores[]={11,12,13,14,15};
```

的“{ }”内共有 5 个值,所以数组 scores 默认有 5 个元素。

另外,在给数组初始化时,“{ }”内数据的个数一定不能超过数组的长度。

(4) 数组元素在定义时确定后,如果在使用时下标超出了定义时的数组长度,称为越界访问,将会产生很严重的错误。例如:

```
#include <stdio.h>
int main()
{
    int a[4]={1,2,3,4};
    int b=12;
    a[4]=6;
    printf("%d,%d\n",b,a[4]);
    return 0;
}
```

的运行结果为

```
6,6
```

可以看出,数组 a 的最大下标元素应为 $a[3]$,代码中的 $a[4]$实际为越界访问,在程序中 $a[4]$的数据覆盖了前面的变量 b,导致最后输出的 b 和 $a[4]$均为 6。

5.1.2 数组元素的引用及基本操作

一个数组变量定义后,就可以对数组元素进行引用。C 语言提供两种方式访问数组:下标方式和指针方式。下面先介绍用下标方式访问数组,指针方式在后面的章节介绍。

1. 数组元素的引用

下标方式引用数组元素的形式如下:

```
数组名[下标]
```

下标可以是整型常量或整型表达式。例如,若定义一个数组 int f[17],则

```
f[2]=f[6]+f[12]-f[3*5]
```

其中,f[3*5]指的就是该数组中的第 16 个元素。

2. 数组元素的基本操作

经常会在程序中成批地处理数组元素。利用下标方式引用数组元素可以在程序运行时动态计算,灵活地控制访问元素。例如,可以通过循环来控制数组中不同元素的引用。

```
int b[10];
int i;
for (i=0;i<10;i++)
    scanf("%d",b[i]);
```

C 语言编译器不会对数组元素的下标表达式作范围检查,所以在编程时要注意不要超界,否则可能会引起意想不到的错误。例如在上面的程序中把条件改为"i<=10",即

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

当 i=10 时,继续执行循环体,这时输出的是内存中 b[9]后面一个字的内容,而不是数组 b 中 10 个元素的值。


5-1.mp4

例 5.1 职工号为 1～10 的 10 位职工的基本工资依次为 412、525、436、352、545、398、560、410、570、380。要求编写程序找出其中的最高工资及最高工资职工的职工号。

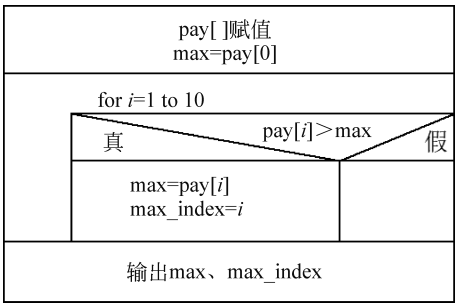

图 5.2 例 5.1 的流程图

分析:定义数组 pay 保存 10 位职工的基本工资,max 保存最大值,处理每一个数组元素,判断该数组元素是否大于 max,如果大于 max,将该数组元素赋值给 max,当所有数组元素都处理完毕后,max 中存储的就是最高工资,当然还需要另一个变量 max_index 来保存最高工资所对应的职工号,而且 max 的初始值应当是第 1 位职工的基本工资。其流程图如图 5.2 所示。

程序代码如下:

```
#include <stdio.h>
int main()
{
    int pay[10]={412,525,436,352,545,398,560,410,570,380};
    int max,max_index,i;
    max=pay[0];
    max_index=0;
```

```
    for (i=1;i<10;i++)
    {
        if (pay[i]>max)
        {
            max=pay[i];
            max_index=i;
        }
    }
    printf( "10 位职工的工资: ");
    for (i=0;i<10;i++)
    {
        printf("%d ",pay[i]);
    }
    printf("\n 最高工资: %d",max);
    printf("\n 该职工的职工号为%d\n",max_index+1);
    return 0;
}
```

程序运行结果如下：

```
10 位职工的工资: 412 525 436 352 545 398 560 410 570 380
最高工资: 570
该职工的职工号为 9
```

思考：如果把该程序改为求出职工的最低工资，应该如何修改呢？

例 5.2　编程用“冒泡法”对 10 个数进行排序。

解题思路：把数组元素按各自值的大小进行整理，数组元素值按从小到大（或从大到小）的顺序重新存放的过程称为数组排序。排序问题是计算机学科中的典型问题，已研制出许多有效的排序算法。冒泡法排序的思想是对数组作多次比较调整遍历，每次遍历是对遍历范围内的相邻两个数作比较和调整，将小的数调到前面，大的数调到后面（设从小到大排序）。定义 int a[10]存放从键盘输入的 10 个数。对数组 a 中的 10 个数用冒泡法排序步骤为，第 1 次遍历是 a[0]与 a[1]比较，如果 a[0]比 a[1]大，则 a[0]与 a[1]互相交换位置；第 2 次是 a[1]与 a[2]比较，如果 a[2]比 a[3]大，则 a[2]与 a[3]互相交换位置……第 9 次是 a[8]与 a[9]进行比较；如果 a[8]比 a[9]大，则 a[8]与 a[9]互相交换位置。第一次遍历结束后，使得数组中的最大数被调整到 a[9]。第 2 次遍历和第 1 次遍历类似，只不过因为第 1 次遍历已经把最大数放到 a[9]中，第 2 次遍历只需要比较 8 次，第 2 次遍历结束时，最大数放于 a[8]中……直到所有的数按从小到大的顺序排列。其流程图如图 5.3 所示。

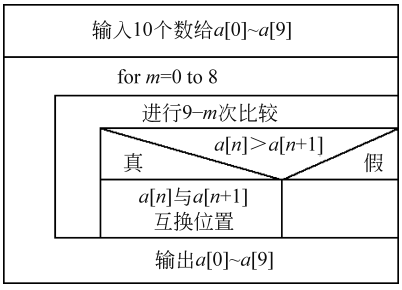

图 5.3　冒泡法排序的流程图（由小到大）

程序代码如下：

```
#include <stdio.h>
int main()
{
    int a[10]={22,33,11,23,45,62,34,62,71,
        30};
```

```
    int m,n,t;
    for (m=0;m<9;m++)
    {
        for (n=0;n<9-m;n++)
            if (a[n]>a[n+1])
            {  t=a[n];
               a[n]=a[n+1];
               a[n+1]=t;
            }
    }
    printf("排序后的 10 个数是\n");
    for (n=0;n<10;n++)
        printf("%d ",a[n]);
    return 0;
}
```

程序运行结果如下：

```
排序后的 10 个数是
11 22 23 30 33 34 45 62 62 71
```

例 5.3 编程输出 Fibonacci 数列前 20 项的值。


5-3.mp4

分析：Fibonacci 数列的特点是开头 2 项都是 1，以后的每一项都等于前两项之和，即 1、1、2、3、5……定义一维数组 f[21]来保存数列的前 20 项，从下标 1 开始使用，开头两项在定义时初始化。从第 3 项开始，每一项都是前两项的和，求出前 20 项，分别存入数组中相应位置。流程图如图 5.4 所示。

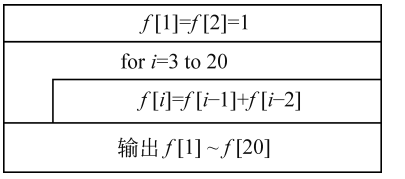

图 5.4 例 5.3 的流程图

程序代码如下：

```
#include <stdio.h>
int main()
{
    int f[21],i;
    f[1]=f[2]=1;
    for (i=3; i<=20; i++)
        f[i]=f[i-1]+f[i-2];
    printf("数列的前 20 项为\n");
    for (i=1; i<=20; i++)
        printf("%2d--->%4d\n",i,f[i]);
    return 0;
}
```

程序运行结果如下：

```
数列的前 20 项为
1--->    1
2--->    1
3--->    2
```

```
4--->    3
5--->    5
6--->    8
7--->   13
8--->   21
9--->   34
10--->  55
11--->  89
12---> 144
13---> 233
14---> 377
15---> 610
16---> 987
17--->1597
18--->2584
19--->4181
20--->6765
```

5.2 二维数组

数组是组织在一起的一批同类型变量，在程序中可以使用数组名和下标来引用其中的任何一个元素。例如，100 个职工的基本工资可以用数组 f[100]来存放，在程序中能方便地用 f[100]来存取第 i 位职工的工资。如果这批职工的工资构成除了基本工资，还有职务工资、津贴等，又该如何来存放这些职工的工资呢。本节引入二维数组的概念，利用它可以在程序中方便地处理任何一位职工的工资。

5.2.1 二维数组的定义、初始化

1. 二维数组的定义

二维数组定义的格式如下：

```
类型  标识符[常量表达式 1][常量表达式 2];
```

其中，“常量表达式 1”用于指定数组第一维的长度，“常量表达式 2”用于指定数组第二维的长度，即每行的元素个数。另外，二维数组对应于一个数学中的矩阵，第一维是行数，第二维是列数。二维数组在实际应用中最为普遍。

例如：

```
int b[6][7];            //6 行 7 列的整型数组
char c[50][50];         //50 行 50 列的字符型数组
```

如图 5.5 所示，数组 a 可以看成一个有 3 个元素的一维数组，每个元素是长度为 4 的一维实型数组。因此数组 a 可分解为 3 个一维数组，即 $a[0]$、$a[1]$、$a[2]$。每个一维数组又含有 4 个元素。例如 $a[0]$数组，含有 $a[0][0]$、$a[0][1]$、$a[0][2]$、$a[0][3]$这 4 个元素，C 语言的二维数组在内存中以先行后列存储的，存放次序是 $a[0][0]$、$a[0][1]$、$a[0][2]$、

$a[0][3]$、…、$a[2][2]$、$a[2][3]$。

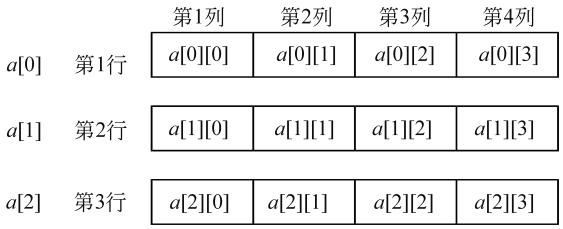

图 5.5　二维数组 *a*

2. 二维数组的初始化

二维数组初始化的方法与一维数组相似，可以用下面的方法对二维数组初始化。

（1）分行给二维数组赋初值，但每行都用“{}”括起来，例如：

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

（2）将所有数据写在“{}”内，按数组排列的顺序对各元素赋初值，例如：

```
int b[4][3]={3,6,2,7,1,9,10,4,11,21,9,4};
```

是按照数组元素下标的排列顺序，依次对 $b[0][0]$、$b[0][1]$、…、$b[3][1]$、$b[3][2]$这 12 个数组元素赋值。

（3）可以对部分数组元素赋值。例如：

```
int[4][3]={{3},{9},{4},{7}};
```

的作用是对各行第一列的元素赋初值，其他元素值自动为 0。

利用初始化的赋值表，可以省略二维数组的行数，由实际数据决定，例如：

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

5.2.2　数组元素的引用及基本操作

二维数组元素的表示形式如下：

```
数组名[下标 1][下标 2]
```

例如 $a[3][4]$表示数组 a 中第 4 行第 5 列的元素。

下标可以是整型表达式，如 $a[3+4][3*4]$。数组元素可以出现在表达式中，也可以被赋值，例如：

```
a[3][1]=9;
a[3][1]=b[4][3];
```

如果定义 a 为 3 行 4 列的数组，它可用的行下标的值最大为 2，列下标的值最大为 3。

如果用 $a[4][5]$ 就超过了数组的范围。

例 5.4 输入和输出一个 3 行 4 列的二维数组。

解题思路：定义二维数组 a[3][4]，注意在输入时，考虑到正常实际应用中没有 0 行和 0 列，因此将数组元素所对应的行数和列数分别加 1。


5-4.mp4

程序代码如下：

```
#include <stdio.h>
int main()
{
    int a[3][4];
    int i,j;
    for (i=0;i<3;i++)
        for (j=0;j<4;j++)
        {
            printf("请输入第 %d 行第 %d 列的元素: ",i+1,j+1);
            scanf("%d",&a[i][j]);
        }
    printf("二维数组的元素是\n");
    for (i=0;i<3;i++)
    {
        for (j=0;j<4;j++)
            printf("%d ",a[i][j]);
        printf("\n");
    }
    return 0;
}
```

程序运行结果略。

例 5.5 编程实现两个矩阵求和运算功能。

解题思路：矩阵是数学中一个重要概念，也是实际工作中处理线性经济模型的重要工具。矩阵由 $m \times n$ 全数排成的 m 行、n 列的数表，简称 $m \times n$ 矩阵。在实际应用中，通常用一个 m 行 n 列的二维数组来表示矩阵。定义两个 3 行 3 列的二维数组 a 和 b，分别通过循环，将两个数组中对应元素相加后赋予第三个二维数组 c，并输出数组 c。流程图如图 5.6 所示。


5-5.mp4

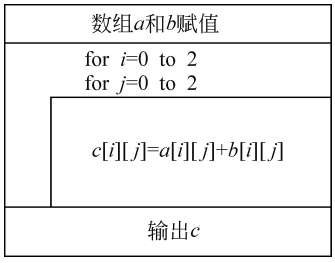

图 5.6 例 5.5 的流程图

程序代码如下：

```
#include <stdio.h>
int main()
{
     int a[3][3]={{6,8,10},{5,3,4},{7,9,10}};
     int b[3][3]={{3,9,1},{12,6,8},{6,21,14}};
     int c[3][3];
     int i,j;
     for (i=0;i<=2;i++)
         for (j=0;j<=2;j++)
```

```
            c[i][j]=a[i][j]+b[i][j];
    printf("两个矩阵的和为\n");
    for (i=0;i<=2;i++)
    {
        for (j=0;j<=2;j++)
            printf("%d\t",c[i][j]);
        printf("\n");
    }
    return 0;
}
```

程序运行结果如下：

```
两个矩阵的和为
9       17      11
17      9       12
13      30      24
```

注意：该程序定义一个新的 3 行 3 列的数组 c，将两个矩阵的和存放到新的数组 c，也可以将两个矩阵对应元素的和直接计算出来。

思考：该题目是直接通过赋初值的方式定义两个矩阵，若将该程序改为从键盘分别输入两个矩阵的值，应该如何修改呢？

例 5.6　用筛选法计算小于 100 的素数。


5-6.mp4

解题思路：利用循环对数组 $a[2]$～$a[100]$赋值 2～100。从 2 开始，从 3 开始往后，筛去所有 2 的倍数（除了 2 本身），标记为 0；再从 3 开始往后，筛去所有 3 的倍数（除了 3 本身），然后是 5 的倍数、7 的倍数等，以此类推，直到筛选到 sqrt(100)＝10 为止。这样，所有未被筛去的数都是素数。流程图如图 5.7 所示。

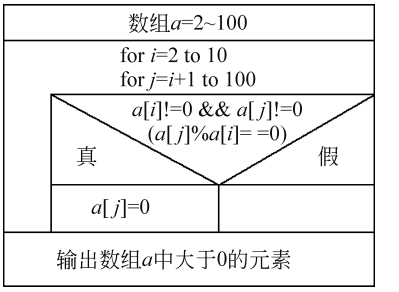

图 5.7　例 5.6 的流程图

程序代码如下：

```
#include <stdio.h>
int main()
{
    int i,j,a[101];
    for (i=2;i<=100;i++)
        a[i]=i;
    for (i=2;i<10;i++)                                  //10=sqrt(100)
        for (j=i+1;j<=100;j++)                          //从 3 开始，到 100 结束
            if (a[i]!=0 && a[j]!=0)                     //已经筛除的不再计算
                if (a[j]%a[i]==0)
                    a[j]=0;
    for (i=2;i<=100;i++)
        if (a[i]!=0)
            printf("%4d",a[i]);
    return 0;
}
```

程序运行结果如下：

```
2  3  5  7  11  13  17  19  23  29 … 89  97
```

例 5.7　编程显示杨辉三角形前 10 行。


5-7.mp4

解题思路：杨辉三角形是由二项展开式的系数所排成的三角形。其特点是三角形边界上的数都是 1，即杨辉三角形每行的第一个数和最后一个数均为 1，除第一行外，每行中间的各数等于上一行位于该数左上方和正上方的两数之和。它表示二项式$(a+b)$的乘方，所得结果的各项依次排列的系数。

例如：

$(a+b)^1=a+b$ 的两项的系数是 1 和 1；

$(a+b)^2=a^2+2ab+b^2$ 的三项系数依次是 1、2、1；

$(a+b)^3=a^3+3a^2b+3ab^2+b^3$ 的 4 项系数依次 1、3、3、1；

…

二项展开式的系数对应“杨辉三角形”上的每一行的数字，定义 $a[10][10]$存储杨辉三角形的各个数，i 作为行的下标，j 作为列的下标，先把杨辉三角形边界上所对应数组中的元素赋值为 1，在计算杨辉三角形内部的元素时，由于各数等于上一行位于该数左上方和正上方的两数之和，即用 $a[i][j]=a[i-1][j-1]+a[i-1][j]$来求出三角形的第 i 行 j 列上的元素，最后输出杨辉三角形上的每个元素的值。该程序分为如下步骤。

(1) 杨辉三角形边界上的元素下标分别为 $a[i][0]$、$a[i][\ i]$，将边界上的数赋初值为 1。

(2) 利用 $a[i][j]=a[i-1][j-1]+a[i-1][j]$，计算杨辉三角形内部每个元素的值，做这一步时要注意并不需要把数组每个元素的值都按这个规则计算出来，对于 10 行的杨辉三角形，从第 3 行开始计算，第 3 行计算 1 个值，下标为 $a[2][1]$，第 4 行计算 2 个值，下标分别为 $a[3][1]$、$a[3][2]$ ……所以要注意外部循环和内部循环的次数。

(3) 将计算过的数组元素按杨辉三角形的模式输出，要注意每行输出元素的个数并非为所有数组元素。

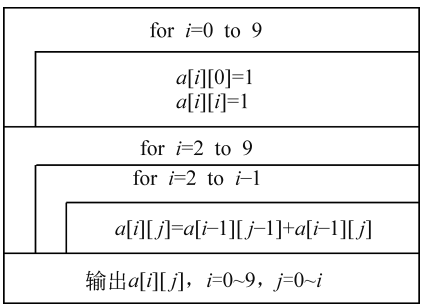

图 5.8　例 5.7 的流程图

程序流程图如图 5.8 所示。

程序代码如下：

```
#include <stdio.h>
int main()
{
    int i,j;
    int a[10][10];
    printf("\n");
    for (i=0;i<10;i++)
```