第 5 章 利用数组处理批量数据

5.1 数组的作用

迄今为止,本书前几章使用的都是属于基本类型(整型、字符型、实型)的数据,它们都是简单的数据类型。对于少量的数据,用以上简单的数据类型处理就可以了,但对数量较大的数据,使用简单的数据类型来处理就不太方便了。例如,一个班有 30 个学生,要分别输入和输出各人的姓名、年龄、成绩等数据,就要定义大量的变量名(如用 s1~s30 代表 30 个学生的学号,age1~age30 代表 30 个学生的年龄等),不仅烦琐,而且这些变量都是孤立的,互无关联,反映不出这些数据的特性(都是同一批学生的学号),难以对它们进行有效、快捷的操作。

为了处理这类问题,人们把同一类性质的数据(如学生的学号)用同一个名字(如 s)来代表,在名字右下角加下标来表示是哪个学生的数据,如用 s_1,s_2,…,s_{30}代表 30 个学生的成绩。这样,这些数据就不是零散的、互不相关的数据,而是一组具有同一属性的数据,这一组数据就成为一个**数组**(array),s 是数组名,下标代表学生的序号。s_{15}代表第 15 个学生的成绩。

由此可知:

(1) 数组是一组有序数据的集合。数组中各数据的排列是有一定规律的,下标代表数据在数组中的序号。

(2) 用一个数组名(如 s)和下标(如 15)来唯一地确定数组中的元素,如 s_{15}就代表第 15 个学生的学号。

(3) 数组中的每个元素都属于同一个数据类型。不能把不同类型的数据(如学生的成绩和学生的性别)放在同一个数组中。

在 C 程序中常根据需要定义数组,并且用循环来对数组中的元素进行操作,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。本章将介绍怎样定义和使用数组。

5.2 怎样定义和引用一维数组

一维数组是最简单的数组，数组元素只有一个下标，用一个数组名和一个下标就能唯一地确定一个数据对象（如用 s_{15} 代表序号为 15 的学生）。除了一维数组以外，还有二维数组（它的元素有两个下标，需要用一个数组名和两个下标才能唯一地确定一个数据对象（如用 $s_{2,3}$ 代表第 2 组第 3 名学生），还有三维数组（它的元素有三个下标，如用 $s_{1,2,4}$ 代表第 1 班第 2 组第 4 名学生）和多维数组（它的元素有多个下标）。它们的概念和用法是相似的。本节先介绍一维数组。

5.2.1 怎样定义一维数组

数组必须先定义后使用，这和定义变量是一样的，计算机不会自动地把一组数据组合成为一个数组。程序设计者必须指定把一批有关联的数据定义为一个数组，并指定数组名、数组中包含数据的个数以及数据的类型。由于用 C 语言的字符无法表示上下角，C 规定用方括号中的数字来表示下标，如用 s[15] 表示 s_{15}，即 s 数组中第 15 个学生的学号。

例如：

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

表示定义了一个整型数组，数组名为 a，此数组有 10 个元素。

定义一维数组的一般形式为

类型符 数组名[常量表达式];

说明：

（1）数组名的命名规则和变量名相同，遵循标识符命名规则。

（2）在定义数组时，需要指定数组中元素的个数，方括号中的常量表达式用来表示元素的个数，即数组长度。例如，定义 a[10]，表示 a 数组有 10 个元素。注意，下标是从 0 开始的，这 10 个元素是：a[0]，a[1]，a[2]，a[3]，a[4]，a[5]，a[6]，a[7]，a[8]，a[9]。请特别注意，按上面的定义，不存在数组元素 a[10]。

（3）上面“常量表达式”中可以包括常量和符号常量，不能包含变量。在主函数中不允许对数组的大小作动态定义，即数组的大小不依赖程序运行过程中变量的值。例如，下面这样定义数组是不对的：

```
int n;
scanf("%d",&n);                    //试图在程序中临时输入数组的大小 n
int a[n];
```

除了 main 函数之外，在其他函数中允许对数组的大小作动态定义，详见第 6 章。

5.2.2 怎样引用一维数组的元素

在 C 程序中只能逐个引用数组元素而不能一次引用整个数组中的全部数据。数组

元素的表示形式为

数组名[下标]

下标可以是整型常量或整型表达式。例如,下面是合法的元素引用:

```
a[2+4],a[2*3],a[7/3]
```

注意:要区分定义数组时用到的“数组名[常量表达式]”和引用数组元素时用到的“数组名[下标]”,由类型符(如 int)起始的是定义数组。例如:

```
int a[10];          //用 int 定义整形数组 a,数组长度为 10,这是定义
b=a[6];             //此处 6 代表元素序号。a[6]代表数组中序号为 6 的元素,这是引用
```

【例 5.1】 给数组元素 a[0]~a[9]赋值为 0~9,然后按逆序输出各元素的值。

解题思路:先定义一个包含 10 个元素的一维数组,然后用循环对各元素赋值,最后用另一循环先后输出全部元素。

编写程序:

```
#include<stdio.h>
int main()
{ int i,a[10];
  for (i=0; i<=9;i++)
    a[i]=i;                    //把循环变量 i 的值赋给下标为 i 的数组元素
  for(i=9;i>=0; i--)
    printf("%d ",a[i]);        //按逆序输出各元素的值
  printf("\n");
  return 0;
}
```

运行结果:

```
9 8 7 6 5 4 3 2 1 0
```

程序分析:第一个循环的作用是把 0 赋给 a[0],把 1 赋给 a[1]……把 9 赋给 a[9]。注意第 2 个循环是怎样实现按逆序输出各元素的。此题算法比较简单,但体现了用循环处理数组的优越性(简单、高效)。

5.2.3　一维数组的初始化

所谓初始化,就是在定义数组时就使数组元素得到初值,这就可以不必再用赋值语句对各元素赋值。对数组元素的初始化可以用以下方法实现。

(1) 在定义数组时对全部数组元素赋初值。例如:

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

将数组元素的初值依次放在一对花括号内。经过上面的定义和初始化之后,a[0]=0,a[1]=1,a[2]=2,a[3]=3,a[4]=4,a[5]=5,a[6]=6,a[7]=7,a[8]=8,a[9]=9。

（2）可以只给一部分元素赋初值。例如：

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

定义 a 数组有 10 个元素，但花括号内只提供 5 个初值，这表示只给前面 5 个元素赋初值，后 5 个元素值自动置为 0。

（3）如果想使一个数组中全部元素值为 0，可以写成

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

或

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

（4）在对全部数组元素赋初值时，由于数据的个数已经确定，因此可以不指定数组长度。例如：

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

可以写成

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

在第二种写法中，花括号中有 5 个数，系统就会据此自动地定义 a 数组的长度为 5。但若数组长度与提供初值的个数不相同，则数组长度不能省略。例如，想定义数组 a 的长度为 10，就不能省略数组长度的定义，否则，系统会默认数组长度为 5。必须写成

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

这样定义数组 a 长度为 10，但只初始化前 5 个元素，后 5 个元素为 0。

说明：在定义数值型数组时，如果指定了数组的长度，且数组长度超过提供的初始化元素值个数，系统会自动把它们初始化为 0（如果是字符型数组，则把它们初始化为'\0'。如果是指针型数组，则初始化为 NULL，即空指针）。

5.2.4 利用一维数组的典型算法——递推与排序

【例 5.2】 用数组来处理求 Fibonacci 数列问题。

解题思路：从例 4.7 已知这是递推问题，例 4.7 用迭代方法，只定义了两个迭代变量 f1 和 f2，程序就可以顺序计算并输出各数。但是这样做不能在内存中保存这些数据。假如想直接输出数列中第 25 个数，是困难的。如果用数组来处理，反而简单了：每个数组元素代表数列中的一个数，按递推方法依次求出各数，并顺序存放在相应的数组元素中。最后顺序输出各元素即可。

编写程序：

```
#include<stdio.h>
int main()
{ int i;
  int f[20]={1,1};                    //数组有 20 个元素，前两个元素为 1,1
```

```
  for(i=2;i<20;i++)
    f[i]=f[i-2]+f[i-1];              //从前两个元素推出当前的元素
  for(i=0;i<20;i++)
  {
    if(i%5==0) printf("\n");        //输出 5 个数后换行
      printf("%12d",f[i]);
  }
  printf("\n");
  return 0;
}
```

运行结果：

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

程序分析：例 4.7 是用循环来处理简单变量，采用迭代方法。本程序没有用迭代，而是用循环来处理数组，把结果存放在数组中。请读者比较二者的异同。从表面上看，两个程序都能正确求出并输出结果，但例 4.7 程序在顺序求出并输出各个数后，不能保存这些数据。而用数组处理时，把每个数据都保存在各数组元素中，如果要单独输出第 10 个数，是很容易的，直接输出 f[9]即可（请思考：为什么不是输出 f[10]，而是 f[9]）。

if 语句用来控制换行，每行输出 5 个数据。

【例 5.3】　输入 10 个数，要求对它们按由小到大的顺序排列。

解题思路：排序是一种重要而常用的算法，在日常生活和用计算机处理问题中经常会遇到。例如，对学生成绩的排序，各地区人口数的排序，各企业产值的排序等。

对一组数据进行排序的方法很多，本例介绍用“**起泡法**”排序。“起泡法”的思路是：将相邻两个数比较，将小的调到前面，见图 5.1。

为简单起见，先分析 6 个数的排序过程。第一次将第 1 个数 9 和第 2 个数 8 比较，由于 9>8，因此将第 1 个数和第 2 个数对调，8 就成为第 1 个数，9 就成为第 2 个数。第二次将第 2 和第 3 个数（9 和 5）比较并对调……如此共进行 5 次，最后得到 8-5-4-2-0-9 的顺序，可以看到，最大的数 9 已“沉底”，成为最下面一个数，而小的数“上升”了。最小的数 0 已向上“浮起”一个位置。经第 1 趟（包括 5 次比较与交换）后，已得到最大的数 9。然后进行第 2 趟比较，对余下的前面 5 个数（8,5,4,2,0）按上述方法进行比较，见图 5.2。经

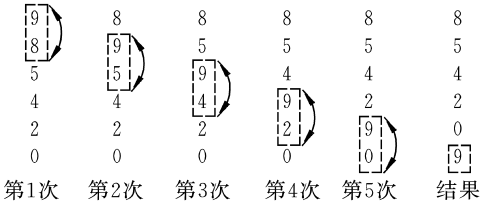

图　5.1

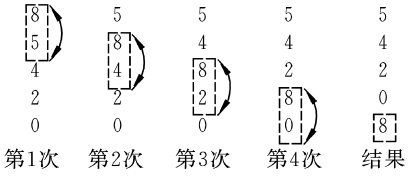

图　5.2

过 4 次比较与交换，得到次大的数 8。如此进行下去，可以推知，对 6 个数要比较 5 趟，才能使 6 个数按大小顺序排列。在第 1 趟中要进行 5 次两个数之间的比较，在第 2 趟中 4 次比较……第 5 趟进行 1 次比较。如果有 n 个数，则要进行 n−1 趟比较。在第 1 趟比较中要进行 n−1 次两两比较，在第 j 趟比较中要进行 n−j 次的两两比较。请读者分析排序的过程，原来 0 是最后一个数，经过第 1 趟的比较与交换，0 上升为第 5 个数（最后第 2 个数），再经过第 2 趟的比较与交换，0 上升为第 4 个数，再经过第 3 趟的比较与交换，0 上升为第 3 个数……每经过一趟比较与交换，最小的数“上升”一位，最后升到第一个数，这如同水底的气泡逐渐冒出水面一样，故称为“冒泡法”或“起泡法”。

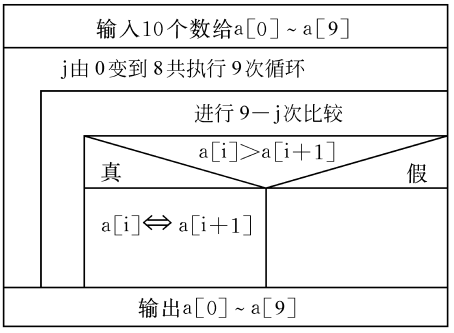

图 5.3

据此画出流程图（见图 5.3，按题意设 n=10）。

编写程序：根据流程图写出以下程序。

```
#include<stdio.h>
int main()
{ int a[10];
  int i,j,t;
  printf("input 10 numbers: \n");
  for (i=0;i<10;i++)
    scanf("%d",&a[i]);
  printf("\n");
  for(j=0;j<9;j++)                          //进行 9 次循环,实现 9 趟比较
    for(i=0;i<9-j;i++)                      //在每一趟中进行 9-j 次比较
      if (a[i]>a[i+1])                      //相邻两个数比较
        {t=a[i];a[i]=a[i+1];a[i+1]=t;}      //如果前数大于后数,使二者对换
  printf("the sorted numbers: \n");
  for(i=0;i<10;i++)
    printf("%d ",a[i]);
  printf("\n");
  return 0;
}
```

运行结果：

```
input 10 numbers:
1 0 4 8 12 65 -76 100 -45 123↙      (从键盘输入 10 个数)
the sorted numbers:
-76 -45 0 1 4 8 12 65 100 123
```

说明：通过此例，着重学习有关排序的算法，理解拿到一个问题之后怎样构思解题的思路。

第 1 章曾介绍过著名计算机科学家沃斯提出的公式：

算法+数据结构=程序

在学习了数组之后，对此公式会有更具体的认识。数组是数据的一种组织形式，或者说是一种数据结构，起泡排序法是一种算法。从本例可以看到，起泡法排序所处理的对象是数组，如果不用数组，难以在 C 程序中实现用起泡法排序。对不同的数据结构，所采用的算法是不同的。因此，一个好的程序应当选择好的算法以及与之适应的数据结构。

除了可以用“起泡法”排序外，还有其他排序方法，在 6.8 节将介绍用“比较交换法”和“选择法”进行排序。请读者分析比较，掌握排序的基本思路。

与一维数组有关的典型算法，除了穷举、递推和排序外，还有查找（从若干数据中快速找到所需的数据），本章的习题 5.9 是用“折半查找法”进行搜索查找，读者可尝试完成该题，或参阅习题解答。

5.3　怎样定义和引用二维数组

只有一维数组是不够的，对有些数据，需要用二维数组来表示。例如有 3 个班，每班 30 人，要表示第 2 班第 5 名学生，需要有两个下标，如 $s_{2,5}$。可以把若干班的学生的有关数据（如学生成绩）组织成一个二维数组。

二维数组常称为**矩阵**（matrix）。把二维数组写成行（column）和列（row）的排列形式，有助于形象化地理解二维数组的逻辑结构。

5.3.1　怎样定义二维数组

先看以下的定义：

```
float a[3][4],b[5][10];                    //用两个方括号表示两个下标
```

定义 a 为 3×4（3 行 4 列）的实型数组，b 为 5×10（5 行 10 列）的实型数组。注意，不能写成

```
float a[3,4],b[5,10];                      //两个下标间不应当用逗号分隔
```

定义二维数组的一般形式为

类型符 数组名[常量表达式][常量表达式];

C 语言对二维数组采用这样的定义方式，使得二维数组可被看作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 a 看作一个一维数组，它有 3 个元素：a[0]，a[1]，a[2]，每个元素又是一个包含 4 个元素的一维数组，见图 5.4。

可以把 a[0]，a[1]，a[2]看作三个一维数组的名字。上面定义的二维数组可以理解为定义了三个一维数组，即相当于

```
float a[0][4],a[1][4],a[2][4];
```

此处把 a[0]，a[1]，a[2]看作一维数组名。C 语言的这种处理方法在数组初始化和用指针表示时显得很方便，这在以后会体会到。

C 语言中，二维数组中元素排列的顺序是按行存放的，即在内存中先顺序存放第一行的元素，再存放第二行的元素。图 5.5 表示对 a[3][4]数组存放的顺序。

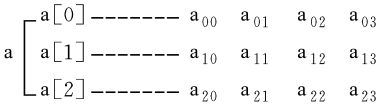

图 5.4

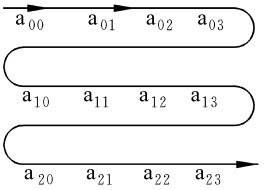

图 5.5

C 语言允许使用多维数组。有了二维数组的基础，再掌握多维数组是不困难的。

5.3.2 怎样引用二维数组的元素

如果已定义：

```
float s[3][30];          //用 s 数组表示学生数据,有 3 个班,每班 30 人
```

若要引用序号为 2 的班中序号为 5 的学生的数据，应表示为 s[2][5]。

二维数组元素的一般形式为

数组名[下标][下标]

下标可以是整常数或整型表达式，如 a[2-1][2*2-1]。不要写成 a[2,3]或a[2-1,2*2-1]形式。

被引用的数组元素可以出现在表达式中，也可以被赋值。例如：

```
b[1][2]=a[2][3]/2
```

注意：在引用数组元素时，下标值应在已定义的数组大小的范围内。下面是常出现的错误：

```
int a[3][4];             //定义 a 为 3×4 的数组
a[3][4]=3;               //想对 a 数组第 3 行第 4 列元素赋值,出错
```

数组 a 可用的行下标范围为 0~2，列下标的范围为 0~3，a[3][4]超过了数组的范围。

5.3.3 二维数组程序举例

【例 5.4】 将一个二维数组 a 的行和列的元素互换（即行列转置）后存到另一个二维数组 b 中。例如：

$$a=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \qquad b=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$$

解题思路：定义两个二维数组：a[2][3]和 b[3][2]，对每一个 a 数组的元素都按以下规律赋给 b 的元素：a[i][j]⇒b[j][i]。用双重循环才能处理所有元素的赋值。

编写程序：

```
#include<stdio.h>
int main()
{
  int a[2][3]={{1,2,3},{4,5,6}};          //定义 a 数组并赋初值
  int b[3][2],i,j;                        //定义 b 数组
  printf("array a: \n");
  for (i=0;i<=1;i++)
  {
    for (j=0;j<=2;j++)
    {
      printf("%5d",a[i][j]);              //输出 a 数组中 i 行 j 列元素
      b[j][i]=a[i][j];              //将 a 数组 i 行 j 列元素赋给 b 数组 j 行 i 列元素
    }
    printf("\n");
  }
  printf("array b: \n");
  for (i=0;i<=2;i++)
  {
    for(j=0;j<=1;j++)
      printf("%5d",b[i][j]);              //输出 b 数组各元素
    printf("\n");
  }
  return 0;
}
```

运行结果：

```
array a:
    1    2    3
    4    5    6
array b:
    1    4
    2    5
    3    6
```

程序分析：在定义数组时可以同时对数组进行初始化。程序第 4 行“int a[2][3]={{1,2,3},{4,5,6}};”的作用是将 1,2,3 赋给 a 数组序号为 0 的行中的 3 个元素(a[0][0],a[0][1],a[0][2]),将 4,5,6 赋给 a 数组序号为 1 的行中的 3 个元素(a[1][0],a[1][1],a[1][2])。

【例 5.5】 有一个单位,下设 3 个组,每组有 4 人,要求找出全体人员中工资最高者以及该职工所在的班组号和该职工在该班组中的序号。

解题思路：先考虑找最大值的算法。读者一定知道在日常生活中“打擂台”是怎样决定最终优胜者的：先找出任一人站在台上,第 2 人上去与之比武,胜者留在台上。再上去

第 3 人，与台上的人（即刚才的得胜者）比武，胜者留台上，败者下台。以后每个人都是与当时留在台上的人比武。直到所有人都上台比过为止，最后留在台上的就是冠军。这就是“打擂台算法”。

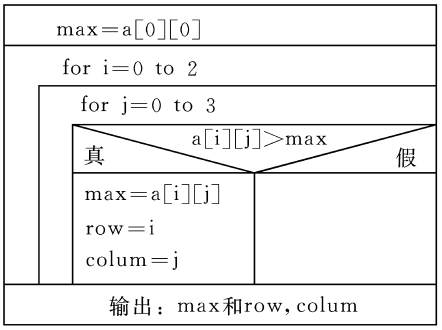

图 5.6

解本题也是用“打擂台算法”。定义一个 3×4 的数组 a，内存放 12 个职工的工资。暂假设最前面的元素 a[0][0]的值最大，把它的值赋给变量 max（max 代表当前最大的数值）。然后将其他各元素依次和 max 比较，如果有大于 max 当前值的，就把该元素的值赋给 max，取代了 max 原来的值。

用 N-S 流程图表示算法，见图 5.6。请读者仔细阅读该流程图。在出现大于 max 的元素时，除了用该元素的值取代 max 原值外，还要把该元素所在的行序号 i 和列序号 j 记录下来，分别存放在变量 row 和 colum 中。全部比完之后，max 的值就是全部元素中的最大值，row 和 colum 的值就是最大元素所在的行序号和列序号。

编写程序：

```
#include<stdio.h>
int main()
{ int i,j,row=0,colum=0,max;
  int a[3][4]={{3123,2145,3211,4321},{5439,3832,6743,4621},{2105,3130,
               5327,3298}};
  max=a[0][0];
  for (i=0;i<=2;i++)
    for (j=0;j<=3;j++)
      if (a[i][j]>max)
        { max=a[i][j];
          row=i;
          colum=j;
        }
  printf("max=%d,group: %d,number: %d\n",max,row+1,colum+1);
  return 0;
}
```

运行结果：

```
max=6743,group: 2,number: 3
```

表示最高工资者是第 2 组的第 3 位职工，工资是 6743 元。

程序分析：

（1）在定义数组时进行初始化，按顺序存入所有职工的工资。按前述的方法，把每个元素的值（即各人工资）先后与 max 的当前值进行比较。注意 max 不是一个固定的值，它的值是不断变化的，它存储的是当时的最高值。当发现某一元素的值大于 max 的当前值