第5章 chapter 5

数　组

本章重点

- 一维数组的定义和引用，顺序排序，选择排序，求一组数中的极值及其位置，逆序存放一组数等常用算法。
- 二维数组的定义、引用和输出，二维数组中元素的表示，二维数组的填充、方阵的转置等常用算法。
- 字符数组和字符串处理函数，不使用字符串处理函数求字符串的有效长度、复制字符串、连接两个字符串、删除字符串中的某种字符等常用算法。

5.1　数组的概念

数组通常用来存放成批的类型相同的数据，数组属于构造数据类型。

5.1.1　引例

【例 5.1】　输入 30 个学生的成绩，求高于平均成绩的人数。

【分析】　按照简单变量和循环结构的方法，求平均成绩的程序如下：

```
#include "stdio.h"
int main()
{   int i,x;
    float s=0,ave;
    for(i=1;i<=30;i++)
    {   scanf("%d",&x);
        s+=x;
    }
    ave=s/30;
    printf("aver=%f\n",ave);
}
```

【说明】

若按这个程序的写法统计高于平均分的人数，则无法实现，因为存放学生成绩的变

量一直使用 x，而 x 是简单变量，只能存放一个学生的成绩，在循环体内输入一个学生的成绩后，就会把前一个学生的成绩覆盖掉。因此，若求完平均成绩后再求高于平均成绩的人数，则必须重新输入这 30 个学生的成绩，不但带来了重复工作，还有可能出现两次输入的成绩不一致的情况，导致统计结果错误。

解决以上问题的方法是：30 个学生的成绩必须有 30 个独立的存放空间，而定义 30 个简单变量的方法会使程序难以处理和推广，所以正确的方法是定义一个数组，使其包含 30 个元素，用来存放 30 个学生的成绩。定义数组时需要指明数组的名字、元素个数和数据类型。一个数组中的所有元素必须都是同一数据类型。

那么，表示 30 个学生的成绩的数组可以定义如下：

```
int x[30];
```

这里的 x 表示数组名，方括号内的 30 表示数组元素的个数，下标为从 0 到 29，即 x[0]，x[1]，…，x[29]，int 表示数组中的所有元素都是整型。求高于平均分的人数的程序如下：

```
#include "stdio.h"
int main()
{   int i,x[30],k=0;
    float s=0,ave;
    for(i=0;i<30;i++)
    {   scanf("%d",&x[i]);
        s+=x[i];
    }
    ave=s/30;
    printf("ave=%f\n",ave);
    for(i=0;i<30;i++)
      if(x[i]>ave) k++;
      printf("num=%d\n",k);
}
```

运行结果：

```
67 68 89 90 75 83 71 56 78 96 49 88 66 65 91 90 56 77 79 91 85 74 81 59 60 69 68 90 98 768
ave=76.166664
num=15
```

对于类型相同且数据量较大的数据，采用数组处理更加便捷。而变量和数组如同生活中的个体与集体的关系，小到个人，大到国家，我们每个人都是集体中的一分子，只有每个人都积极发挥自己的能力，国家才能爆发出无穷的力量，任何一个集体的成功都离不开每个个体的奉献。

5.1.2 数组的概念

数组是相同类型的数据组成的序列，用一个共同的名字表示，数组中数的顺序可以

由它的下标表示。有一个下标的数组称为一维数组，有两个下标的数组称为二维数组；C语言用字符数组表示字符串。

数组在内存中占据一片连续的存储单元。若有定义

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

则表示数组 a 中包括 10 个元素，分别为 a[0]～a[9]。这 10 个元素在内存中是连续存放的，如图 5.1 所示，只要找到 a[0]，后面的元素地址就可以计算出来。C 语言中，数组名代表数组的首地址，即 a[0]的地址。

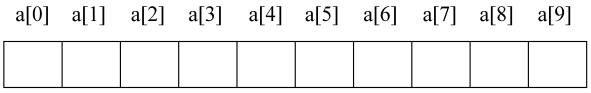

图 5.1　数组 a 的存储分配

C 语言中，数组的下标从 0 开始，例如定义 int a[10]时，最大可用下标为 9，C 语言编译器不对下标越界进行检查，因此程序员必须自己保证下标引用的正确。

5.2　一 维 数 组

5.2.1　一维数组的定义和引用

1. 一维数组的定义

数组必须先定义、后使用，定义一维数组的一般形式为

类型标识符　数组名[数组大小]

例如：

```
int x[100],z[10];
float a[20];
```

一般形式中，“[ ]”内的“数组大小”表示数组中元素的个数，也称数组长度，它必须是整型常量表达式，不可以包含变量。例如，“int n=10; int a[n];”是不合法的。

2. 一维数组元素的引用

一维数组元素引用的一般形式为

数组名[下标]

此处“[]”内的“下标”必须是整型表达式，可以只是一个常量、变量，也可以是表达式。例如 a[1]、a[i]、a[9－i](i 为整型)都是合法的数组元素引用，数组元素称为下标变量。

5.2.2 一维数组元素的赋值

1. 初始化

数组在定义之后，若不给其赋值，则其值是无法预料的不确定值，可以使数组在程序运行之前初始化，即在编译期间使之得到初值。

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，对数组元素赋初值。

例如：

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

(2) 只给一部分元素赋值。

例如：

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

表示只给前 5 个元素赋初值，后 5 个元素自动赋 0 值。

(3) 若对 static 数组不赋初值，则系统会对所有元素自动赋 0 值。

如果想使数组 a 中的全部元素值为 0，则可以这样定义数组：

“static int a[5];”等价于“int a[5]={0};”。

(4) 在对全部数组元素赋初值时，可以不指定数组长度。

例如：

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

和下面的写法是等价的：

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

2. 数组元素的输入

数组元素的处理是和循环分不开的，输入和输出都需要结合循环结构进行。如例 5.1 中的第 4 行和第 5 行就是用循环输入数组中的所有元素的。

3. 数组元素的赋值

如果数组中待赋的数组元素的值是有规律的，那么可以使用循环语句直接给数组元素赋值，不必一个一个地输入。

【例 5.2】 数组元素的赋值。

程序如下：

```
#include "stdio.h"
int main()
{   int i,a[10];
```

```
    for(i=0;i<10;i++)
        a[i]=i+1;
    for(i=0;i<10;i++)
        printf("%5d",a[i]);
}
```

运行结果：

```
1    2    3    4    5    6    7    8    9    10
```

4. 产生随机数

如果不要求数组中的数是有规律的，则可以用随机函数产生一组随机数并赋值给数组中的元素，产生随机数的函数为 rand，包含在 stdlib.h 头文件中。使用时，需要在主函数之前使用文件包含命令 #include "stdlib.h"或 #include <stdlib.h>，该函数的一般形式为 rand()。

5.2.3 一维数组常用算法举例

【例 5.3】 求 10 个数中的最大值和最小值。

【分析】 求极值的算法在第 4 章的循环部分已经介绍过，那时是使用简单变量完成的，现在用数组完成，请读者体会一下使用数组的方便之处。

程序如下：

```
#include "stdio.h"
int main()
{   int i,a[10],min,max;
    for(i=0;i<=9;i++)
        scanf("%d",&a[i]);
    max=min=a[0];
    for(i=1;i<10;i++)
    {   if(a[i]<min)  min=a[i];
        if(a[i]>max)  max=a[i];
    }
    printf("max=%d,min=%d\n", max,min);
}
```

运行结果：

```
34 56 78 31 46 79 12 34 57 22
max=79,min=12
```

【例 5.4】 求 10 个数中的最小值及其位置，并将其换到第 1 个数的位置。

【分析】 当待处理的数据涉及顺序、位置等关系时，用数组存放数据比较方便。找最小值及其位置的算法如下，对应的结构化流程图如图 5.2 所示。

(1) 定义一个包含 10 个整型元素的一维数组，即 int a[10];，然后输入 10 个数。

(2) 找最小值及其位置。

① 用 k 表示最小值的下标(位置),则 a[k]就表示最小值,设 k 的初值为 0,即首先假设 a[0]最小。

② 找最小值:数组中从 a[1]到 a[9]的每个数都和最小值 a[k]比较,如果有某个 a[i]<a[k],则表示 a[i]为当前最小值,需要将 i 的值赋给 k。循环完成之后,a[k]就是找到的最小值,k 表示最小值的位置。

(3) 将 a[k]和 a[0]互换。

程序如下:

```
#include "stdio.h"
int main()
{   int i,t,a[10],k;
    for(i=0;i<=9;i++)
        scanf("%d",&a[i]);
    k=0;
    for(i=1;i<10;i++)
        if(a[i]<a[k])  k=i;
    if(k!=0)            /* 如果 k 为 0 就不用换了 */
      {t=a[0]; a[0]=a[k]; a[k]=t;}
    printf("min number is:%d\n",a[0]);
    printf("the position is:%d\n", k);
}
```

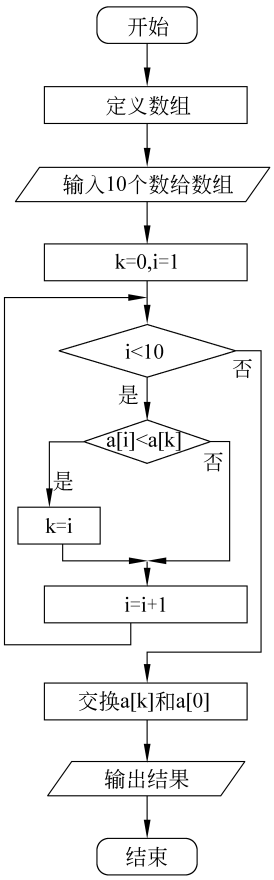

图 5.2 求最小值并将其换到第 1 个数的位置

【说明】 这个程序完成的是从 10 个数中找到最小值,然后将其换到 a[0]的位置,按照同样的做法,可以从 a[1]到 a[9]这 9 个数中找到最小值并将其换到 a[1]的位置,再从 a[2]到 a[9]中找到最小值并将其换到 a[2]的位置,以此类推,最后从 a[8]到 a[9]中找到最小值换到 a[8]的位置,剩下的 a[9]就是这 10 个数中的最大值。这样做完之后,就完成了将 10 个数按照由小到大的顺序排序。这种排序的方法称为选择排序法。简单地说,选择排序法就是重复 9 次求最小值并将其换到相应位置的操作。对 n 个数进行排序的算法描述如下。

(1) 产生数组。

(2) 对 i=0,1,…,n−1 做

① k=i;

② 对 j=i+1,…,n 做

如果 a[j]<a[k],则 k=j;

③ 交换 a[k]和 a[i]。

【例 5.5】 将一维数组中的 n(n≤50)个数按逆序存放。

【分析】 当处理的数据个数 n 不确定时,可以先定义一个足够大的数组,然后输入 n,最后输入 n 个数。逆序存放数组可以采用图 5.3 所示的方法。

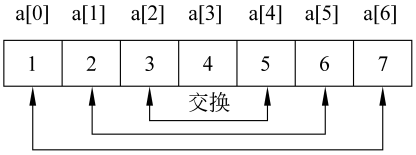

图 5.3 逆序存放数组的方法

当 n=7 时,a[0]和 a[6]交换,a[1]和 a[5]交换,a[2]和 a[4],a[3]不变或自己和自己交换。进行循环时,i 从 0 循环到(n−1)/2,a[i]和 a[n−1−i]交换。

程序如下:

```
#include "stdio.h"
int main()
{   int a[50],t,n,i;
    scanf("%d",&n);                          /*输入元素个数 n*/
    for (i=0;i<n;i++)
        {   scanf("%d",&a[i]);               /*输入 n 个数*/
            printf("%5d",a[i]);}
    printf("\n");
    for (i=0;i<=(n-1)/2;i++)
        {   t=a[i];
            a[i]=a[n-1-i];
            a[n-1-i]=t;
        }
    for(i=0;i<n;i++)
    printf("%5d",a[i]);
}
```

运行结果:

```
7 1 2 3 4 5 6 7
    1    2    3    4    5    6    7
    7    6    5    4    3    2    1
```

【例 5.6】 用二分查找法查找一个数是否在一个有序数组中。

【分析】 二分查找法也称折半查找法,用来在一组数中查找是否存在某个已知的数,前提条件是这组数必须是有序的,即这组数是按升序或降序排序的。设数组 a 中的 n 个数是按升序排序的,查找 key 是否在数组 a 中的算法如下。

(1) 设开始查找的范围为整个数组,则下标为从 0 到 n−1,用 low 和 high 表示查找范围的端点,即 low=0,high=n−1。

(2) 取查找范围的中点 mid=(low+high)/2。

(3) 将待查找的数 key 和中间的数 a[mid]进行比较,如果 key=a[mid],则表示找到了,输出 key 在数组中的位置 mid,结束循环;如果 key<a[mid],则表示 key 一定在数组前半部分,即可把查找范围缩小一半,可将查找范围的右端点变为中间元素的前面一个

元素的位置，即 high＝mid－1。如果 key＞a[mid]，则表示 key 一定在数组的后半部分，可将 low 变为 mid＋1，然后在新的范围内用同样的方法继续查找，直到 low＞high 为止。如果待查找的数不在数组中，则输出没有找到的信息。流程图如图 5.4 所示。

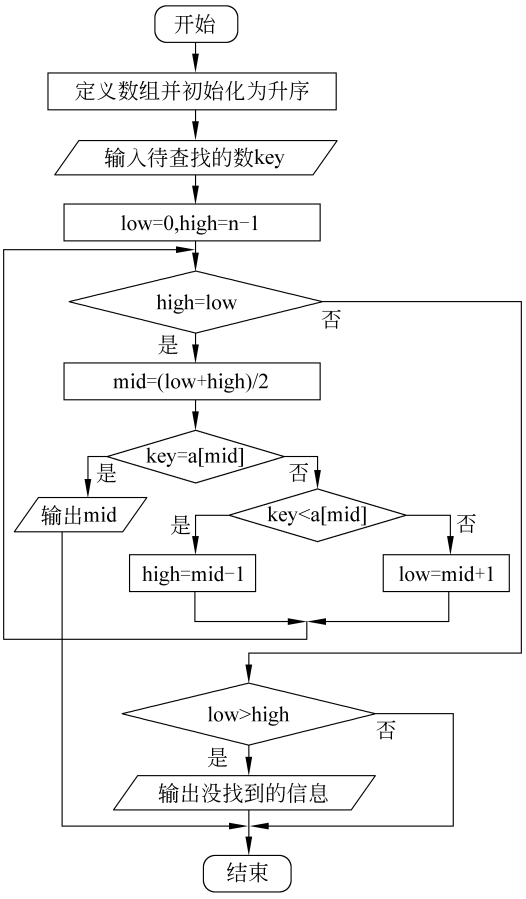

图 5.4　二分查找法流程图

程序如下：

```
#include "stdio.h"
int main()
{   int n=12,a[]={1,3,5,7,9,12,15,19,24,27,38,66},key,mid,low,high;
    printf("input the number:\n");
    scanf("%d",&key);
    low=0,high=n-1;
    while(low<=high)
    {   mid=(low+high)/2;
        if(key==a[mid])
        {   printf("%d\n",mid);
            break;
        }
```

```
        else if(key<a[mid])
            high=mid-1;
        else low=mid+1;
    }
    if(low>high)
    printf("not found\n");
}
```

运行结果：

```
input the number:
35
not found
input the number:
27
9
```

使用一维数组还可以处理很多事情，例如在成绩处理中求全班学生的平均分、最高分、最低分，排名次，查找是否存在某分数及某人，插入学生成绩，删除学生成绩等。如果需要处理一个学生的 4 门课的成绩，则可以定义一个包含 4 个元素的一维数组，而要想表示 30 个学生的 4 门课的成绩，就可以使用二维数组。

定义：

```
int a[30][4];
```

这里的 a[i]表示第 i 个学生，a[i][0]、a[i][1]、a[i][2]和 a[i][3]分别表示第 i 个学生的 4 门课的成绩。

5.3 二维数组

5.3.1 二维数组的定义

1. 二维数组的形式

二维数组的一般形式为

类型标识符　数组名[整型常量表达式][整型常量表达式]

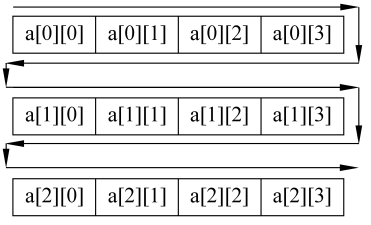

图 5.5　二维数组的存储顺序示意

“[]”内的“整型常量表达式”表示数组每维的大小或每维的元素个数。

例如，定义“ int a[3][4]”，表示定义数组 a 包含 3 行 4 列，共 12 个元素，这 12 个元素在内存中占据一片连续的存储单元，存放的顺序是先行后列，即先放第 0 行，再放第 1 行，最后放第 2 行，如图 5.5 所示。

图 5.5 中的每行都可以看成一个一维数组，3 个一维数组的名字分别为 a[0]、a[1]、a[2]，每个一维数组中包含 4 个元素，即二维数组可以看成特殊的一维数组，特殊之处在于每个数组元素还是一个一维数组。

2. 二维数组的引用

二维数组的引用形式为

数组名[下标][下标]

注意：下标可以是整型表达式，但应在已定义的数组范围内，不能整体引用数组，只能一个元素一个元素地引用。

3. 多维数组的定义

多维数组的定义方式与二维数组类似。以下定义了两个三维数组：

```
int x[3][4][2];
float y[4][1][3];
```

5.3.2 二维数组的初始化

(1) 分行给二维数组赋初值。

例如：

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

(2) 可以将所有数据写在花括号内，按数组排列的顺序为各元素赋初值。

例如：

```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};赋值的结果和 1 中相同
```

(3) 可以为部分元素赋初值，对于没有赋值的元素，系统自动赋 0 值。

例如：

```
int a[3][4]={{1},{5},{9}};
int a[3][4]={{1},{0,6},{0,0,11}};
int a[3][4]={{1},{5,6}};
```

(4) 如果为全部元素赋初值，则定义数组时可以不指定第一维的长度，但不能省略第二维的长度。

例如：

```
int  a[3][3]={1,0,3,4,0,0,0,0,9}
int  a[ ][3]={1,0,3,4,0,0,0,0,9}
int  a[ ][3]={{1,0,3},{4},{0,0,9}}
```