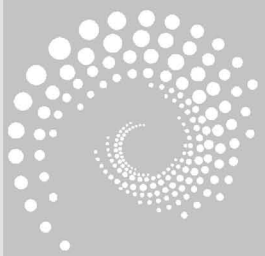
第5章

用数组处理批量数据

CHAPTER 5



不同于基本数据类型（int、float、char 等），数组是一种构造数据类型，是存储具有相同类型元素的集合。这些数组元素在内存中连续存放，将数组和循环结合起来，可以高效方便地处理批量数据。

学习目标

- 掌握一维数组的定义、存储、初始化与基本操作。
- 掌握二维数组的定义、存储、初始化与基本操作。
- 理解字符串与字符数组的联系和区别。

5.1 认识数组

数组是程序设计中非常重要的一种数据结构，常用于存储相同类型的数据元素，并按照一定的顺序进行存储。数组分为一维数组和多维数组，最常见的多维数组是二维数组。

一维数组是最简单的数组。例如，要存储某个班级 10 个学生的数学成绩（float 型），如果使用普通变量，每个变量只能存储一个成绩，需要定义 10 个 float 型的变量 a1、a2、a3、…、a10。可以将这 10 个数据类型相同的变量组成一个一维数组 a，每个变量是数组中的一个元素。通过数组名 a 和下标（从 0 开始）来唯一地确定数组中的元素，如 a[0]表示第 1 个学生的成绩，a[1]表示第 2 个学生的成绩，以此类推。

【例 5-1】计算某个班级 10 个学生的数学成绩平均分，并输出所有高于平均分的成绩。

分析：采用前面学习的编程方法。如果定义 10 个 float 型的变量 a1、a2、a3、…、a10 来保存 10 个学生成绩，则无法使用循环结构实现。此外，如果学生人数增加到 100 个、1000 个或更多时，定义如此多的变量也不现实。如果使用循环结构，循环体中只能使用一个变量保存输入的学生成绩，无法保存 10 个成绩，也无法实现将 10 个成绩与平均分进行比较的功能。因此，必须使用数组来解决这个问题。首先定义一个一维数组来保存 10 个学生的成绩，然后在循环体中通过不同的下标访问不同的数组元素。

示例程序如下。

```
#include <stdio.h>
#define N 10
int main()
{   int i;
    float avg=0;                        /*保存平均分*/
    float a[N];                         /*定义一个一维数组，用来存放 10 个成绩*/
    printf("请输入%d 个成绩：", N);
    for(i=0; i<N; i++)                  /*循环 10 次，每次读入 1 个成绩，并累加*/
    {   scanf("%f", &a[i]);
        avg+=a[i];
    }
    avg=avg/10;                         /*求平均*/
    printf("数学成绩平均分为：%.2f\n", avg);
    printf("高于平均分的成绩有：");
    for(i=0; i<N; i++)                  /*输出所有大于平均分的学生成绩*/
        if(a[i]>avg)
            printf("%.2f ", a[i]);
    printf("\n");
    return 0;
}
```

程序执行结果如下。

```
请输入 10 个成绩：90 78 86 85 69 70 55 89 65 95
```

```
数学成绩平均分为：78.20
高于平均分的成绩有：90.00 86.00 85.00 89.00 95.00
```

5.2　一维数组

一维数组是最简单的数组，通过数组名和一个下标来唯一地确定数组中的元素。

5.2.1　一维数组的定义和引用

与变量一样，数组在使用前必须先定义，即通知计算机数组中有多少元素，是什么数据类型，否则计算机不会自动地把一批数据作为数组处理。

1．一维数组的定义

定义一维数组的一般形式如下。

```
数据类型 数组名[数组长度];
```

例如，下面的一维数组定义语句。

```
int a[5];          /*定义了一个整型数组，数组名为 a，包含 5 个数组元素*/
float b[10];       /*定义了一个 float 型数组，数组名为 b，包含 10 个数组元素*/
char c[80];        /*定义了一个 char 型数组，数组名为 c，包含 80 个数组元素*/
```

一维数组定义时需注意以下几点。

（1）数据类型指定了数组中每个元素的数据类型。

（2）数组名的命名规则与变量名相同，应遵循标识符命名规则。

（3）数组的大小是在编译时确定的，因此必须指明数组长度，数组长度只能是一个整型常量表达式，不能包含变量，且方括号不可省略。例如，下面的一维数组定义是错误的。

```
int n;
scanf("%d", &n);
int a[n];              /*数组 a 的定义中，数组长度不能出现变量*/
```

2．一维数组元素的引用

C 语言规定对于数值型数组，只能逐个引用数组中的元素，而不能一次引用整个数组全部元素的值。一维数组元素的引用需要指定下标，引用形式如下。

```
数组名[下标];
```

引用一维数组元素的下标可以是整型常量、变量或表达式。引用一维数组元素时要注意以下几点。

（1）下标从 0 开始，它的合理取值范围是[0，数组长度−1]；

（2）程序运行时系统并不会自动检查数组下标是否越界，因此编程人员一定要保证数

组下标不能越界。

例如，有以下的一维数组定义语句。

```
int a[10], i=3;
```

则数组 a 中的元素分别是 a[0]、a[1]、a[2]、…、a[9]。需要注意的是，数组 a 中没有元素 a[10]，但是在程序中 a[10]、a[11]也是可以引用的，只是它们的数据无法确定。a[2]、a[i]、a[i+2]、a[i*2]、a[8/2]都是对一维数组元素的正确引用，但 a[−2]、a[1.5]、a[i*2.3]、a[1][2]都是对一维数组的错误引用。

5.2.2 一维数组的存储和初始化

定义一维数组后，系统将按数组的类型和长度在内存中开辟一片连续的存储单元，首地址用数组名表示，占用的字节数由数据类型和数组长度决定。

1. 一维数组的存储

例如，有以下的一维数组定义语句。

```
int a[5];
```

系统将在内存中开辟 5 个连续的整型数据存储单元存放这 5 个数组元素，它们在内存中按下标递增的顺序连续存放，每个元素占用 4 字节，共占用 20 字节。数组名 a 表示该连续存储单元的首地址，也就是数组中第一个元素 a[0]的地址，是一个地址常量，如图 5-1 所示。

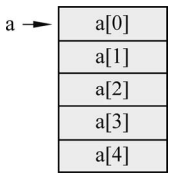


图 5-1　一维数组内存示意图

2. 一维数组的初始化

对一维数组元素赋值有两种方式。一种是使用赋值语句或 scanf()函数在程序运行时对数组元素赋值。另一种是在数组定义时给数组元素赋初值，这种方式称为数组的初始化，初始化在程序编译时进行。若定义一维数组时没有对数组进行初始化，则数组元素的值为随机数。对一维数组元素的初始化可以通过以下方法实现。

（1）在定义一维数组时对所有数组元素赋初值，将数组元素的全部初值依次放在一对花括号内。例如，有以下的一维数组定义语句。

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

则 a[0]=1，a[1]=2，a[2]=3，a[3]=4，a[4]=5。若数组长度小于初值的个数，则编译时出错。例如，有以下的一维定义语句，程序编译时将出错。

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

（2）在定义一维数组时给一部分数组元素赋初值，在一对花括号内只给出部分数组元素的初值，此时其余元素值默认为 0。例如，有以下的一维数组定义语句。

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

则 a[0]=1，a[1]=2，a[2]=3，a[3]=0，a[4]=0，因此若要将一个数组中全部元素都初始化为 0，可以按下面的形式定义一维数组。

```
int a[5]={0};
```

（3）在定义一维数组时对所有数组元素赋初值，由于初值的个数与数组元素的个数一一对应，因此可以不指定数组长度。例如，有以下的一维数组定义语句。

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

花括号中有 5 个初始值，系统就会据此确定数组 a 的长度为 5。

【例 5-2】 一维数组的初始化示例

示例程序如下。

```
#include <stdio.h>
#define N 5
int main()
{   int i, a[N], b[N]={10,-2};
    for(i=0; i<N; i++)
       printf("%d", a[i]);
    printf("\n");
    for(i=0; i<N; i++)
       printf("%d", b[i]);
    return 0;
}
```

程序执行结果如下。

```
-858993460  2  -858993460  1990005090  -858993460
10  -2  0  0  0
```

程序执行结果分析如下。

（1）数组 a 在定义时没有对其初始化，所以数组 a 的 5 个元素值均为随机数；

（2）数组 b 在定义时进行了初始化，b[0]初始化为 10，b[1]初始化为−2，后 3 个元素被默认地初始化为 0。

5.2.3　一维数组应用示例

【例 5-3】输入 10 个互不相同的整数保存在一维数组中，输出最大值和最大值的下标。

分析：用 max_index 表示数组 a 中最大元素所在位置的下标。首先假定第一个元素是最大值，即 max_index=0，然后将数组中剩下的元素逐个与下标为 max_index 的元素进行比

较（打擂台），如果当前元素大于下标为 max_index 的元素，那么更新 max_index 为当前元素的下标，从而保证 max_index 中始终存放的是当前比较过的元素中最大值的下标。当所有元素都比较过以后，最大值是 a[max_index]，最大值的下标是 max_index。

示例程序如下。

```
#include <stdio.h>
#define N 10
int main()
{   int a[N], i=0, max_index;
    printf("请输入%d 个整数： ", N);
    for(i=0; i<N; i++)          /*读入数组 a 各元素的值*/
       scanf("%d", &a[i]);
    max_index=0;                /*寻找并记录数组中最大值元素的下标*/
    for(i=1; i < N; i++)
       if(a[max_index]<a[i])    /*max_index 保存数组中最大值元素的下标*/
          max_index=i;
    printf("最大数是：a[%d]=%d\n", max_index, a[max_index]);
    return 0;
}
```

程序执行结果如下。

```
请输入 10 个整数： 56  78  89  12  40  10  2  -30  5  23
最大数是：a[2]=89
```

【例 5-4】 在整型数组 a 中查找 x 第一次出现的位置，如果找到了，输出相应的下标，否则，输出 x 不存在。

分析：设置标志变量 flag 来存储下标，初始化为−1 表示没有找到，然后从第一个元素开始依次将数组元素与 x 进行比较，如果相等，表示找到 x，将当前元素下标赋值给变量 flag，然后结束比较，跳出循环。这种查找方法称为顺序查找，可以在无序数组中查找变量 x 是否存在。

示例程序如下。

```
#include <stdio.h>
#define N 10
int main()
{   int i, flag=-1, x;              /*标志变量 flag 初始化为-1 表示没有找到*/
    int a[N]={22,19,36,80,98,12,20,55,-8,16};     /*定义并初始化数组*/
    for(i=0; i<N; i++)
       printf("%d", a[i]);
    printf("\n");
    printf("请输入 x：");
    scanf("%d", &x);
    for(i=0; i<N; i++)
       if(a[i]==x)                  /*如果在数组 a 中找到了 x*/
       {   flag=i;                  /*将当前元素下标赋值给 flag*/
```

```
            break;                          /*跳出循环，结束查找*/
        }
    if(flag!=-1)
        printf("%d 在数组中第一次出现的位置是%d\n", x, flag);
    else
        printf("%d 在数组中不存在", x);
    return 0;
}
```

输入不同的整数 x 的程序执行结果如表 5-1 所示。

表 5-1　输入不同的整数 x 的程序执行结果

序号	不同取值的 x	程序执行结果
第一组	数组中的元素	22 19 36 80 98 12 20 55 −8 16 请输入 x：16 16 在数组中第一次出现的位置是 9
第二组	数组之外的元素	22 19 36 80 98 12 20 55 −8 16 请输入 x：0 0 在数组中不存在

在示例程序中没有考虑数组中有多个重复的 x 存在的情况，若要求找出 x 出现的所有位置，程序将如何改写，请读者思考。

【例 5-5】用数组改写例 4-19，求斐波那契（Fibonacci）数列的前 20 个数，每行输出 5 个数。

分析：定义一个数组 fib 来计算并存放斐波那契数列的前 20 个数，有如下关系成立。

$$fib[0]=fib[1]=1$$

$$fib[i]=fib[i-1]+fib[i-2] \quad (2\leqslant i\leqslant 19)$$

示例程序如下，程序执行结果与例 4-19 完全一致。

```
#include <stdio.h>
#define N 20
int main()
{   int i;
    int fib[N]={1,1};                   /*数组定义并初始化前 2 个数*/
    for(i=2; i<N; i++)                  /*计算斐波那契数列其余的 18 个数*/
        fib[i]=fib[i-1]+fib[i-2];
    for(i=0; i<N; i++)                  /*输出斐波那契数列*/
     {  printf("%10d", fib[i]);
        if((i+1)%5==0)
            printf("\n");
     }
    return 0;
}
```

【例 5-6】 输入一个正整数 n（1<n≤10），再输入 n 个整数，用冒泡法将它们从小到大排序后输出。

分析：给定 n 的取值范围 1<n≤10，说明数组长度为 10。冒泡法是在解决排序问题时的一种常用方法。它的基本思路是：依次将数组中相邻两个元素进行比较，如果前一个元素比后一个元素大，则交换。在比较完数组中所有的元素后，最大的数就“沉到”最后的位置，完成了一趟排序。例如，6 个数的第一趟冒泡排序过程如下。

初始排列：　9　8　5　4　2　0
第一次冒泡比较：　8 ⇆ 9　5　4　2　0
第二次冒泡比较：　8　5 ⇆ 9　4　2　0
第三次冒泡比较：　8　5　4 ⇆ 9　2　0
第四次冒泡比较：　8　5　4　2 ⇆ 9　0
第五次冒泡比较：　8　5　4　2　0 ⇆ 9

经过第一趟（共 5 次比较和交换）排序后，最大的数 9 已经到了最后的位置。接下来进行第二趟排序，即对余下的前 5 个数（8，5，4，2，0）按同样的方法进行相邻元素比较和交换，会把最大的数 8 交换到这 5 个数最后的位置（5，4，2，0，8）。以此类推，第一趟进行了 5 次比较和交换；第二趟进行了 4 次比较和交换；…；第五趟只需要进行 1 次比较和交换。因此，如果有 n 个数要排序，则总共要进行 n−1 趟，每趟要进行 n−i 次相邻两数的比较。

示例程序如下。

```
#include <stdio.h>
#define N 10
int main()
{   int i, j, t, n;
    int a[N];
    printf("请输入要进行排序的整数个数(小于等于%d)：", N);
    scanf("%d", &n);
    printf("请输入%d 个整数：", n);
    for(i=0; i<n; i++)                    /*输入数组元素*/
       scanf("%d", &a[i]);
    for(i=0; i<n-1; i++)                  /*对数组 a 中的 n 个元素冒泡法排序*/
       for(j=0; j<n-i-1; j++)             /*共 n-1 趟，每趟比较 n-i 次*/
          if(a[j]>a[j+1])
          {   t=a[j];
              a[j]=a[j+1];
              a[j+1]=t;
           }
    printf("由小到大的排序结果是：");
    for(i=0; i<n; i++)
       printf("%d", a[i]);
    printf("\n");
    return 0;
}
```

程序执行结果如下。

```
请输入要进行排序的整数个数(小于等于 10)：5
请输入 5 个整数：99 35 24 103 22
由小到大的排序结果是：22  24  35  99  103
```

【例 5-7】 输入一个正整数 n（1<n≤10），再输入 n 个整数，用选择法将它们从小到大排序后输出。

分析：选择法排序的核心思想是将 n 个数进行 n−1 趟循环选择。每趟循环都在参与选择的数中选择出最小的数据，并将它与本趟参与选择的第一个数进行交换；以此类推，当进行完 n−1 趟循环选择后，n 个数就按照从小到大的顺序排列好了。

选择排序的算法步骤如下。

第一趟：在未排序的 n 个数（a[0]～a[n−1]）中找到最小的元素，将它与 a[0]交换；

第二趟：在剩下未排序的 n−1 个数（a[1]～a[n−1]）中找到最小的元素，将它与 a[1]交换；

……

第 n−1 趟：在剩下未排序的 2 个数（a[n−2]～a[n−1]）中找到最小的元素，将它与 a[n−2]交换。

示例程序如下。

```
#include <stdio.h>
#define N 10
int main()
{   int i, index, k, n, temp;
    int a[N];
    printf("请输入要进行排序的整数个数(小于等于%d)：", N);
    scanf("%d", &n);
    printf("请输入%d 个整数：", n);
    for(i=0; i<n; i++)                          /*输入数组元素*/
        scanf("%d", &a[i]);
    for(k=0; k<n-1; k++)                        /*对数组 a 中的 n 个元素排序*/
    {   /*在每一趟排序前，指定最小元素的初始下标 index 为未排序的第一个元素*/
        index=k;
        for(i=k+1; i<n; i++)                    /*在未排序数中查找到最小元素*/
            if(a[i]<a[index])
                index=i;
        temp=a[index];            /*将最小元素 a[index]与第一个元素 a[k]互换*/
        a[index]=a[k];
        a[k]=temp;
    }
    printf("由小到大的排序结果是：");
    for(i=0; i<n; i++)
        printf("%d", a[i]);
    printf("\n");
    return 0;
}
```

程序执行结果如下。

```
请输入要进行排序的整数个数(小于等于 10)：5
请输入 5 个整数：99 35 24 103 22
由小到大的排序结果是：22  24  35  99  103
```

【例 5-8】 在有序整型数组 a 中插入整数 x，要求插入后的数组仍保持有序。

分析：进行插入数据操作的前提是数组足够大，即数组长度 N 大于数组中有效数据个数 len，插入完成后，数组中有效数据个数 len 加 1。在插入 x 之前，首先要明确插入的位置，即第一个比 x 大的元素下标 index。进行插入操作之前，需要将从下标为 index 开始的元素后移，但是需要注意的是，要从数组最后一个元素开始后移操作，一直到下标为 index 的这个元素为止。然后将 x 赋值给下标为 index 的元素。

例如，假定数组 a 长度为 10，有效数据个数 len=7，待插入整数 x=40，则插入位置 index=4，移动数组元素的过程如图 5-2 所示，其中①、②和③是移动的顺序。

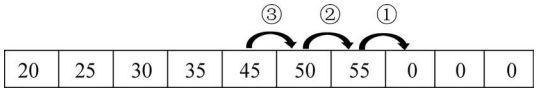


图 5-2 插入 40 的数组元素移动示意图

示例程序如下。

```
#include <stdio.h>
#define N 10
int main()
{   int a[N]={20,25,30,35,45,50,55}, len=7, index=0, x, i ;
    printf("原始数组：");
    for(i=0; i<len; i++)
        printf("%5d", a[i]);
    if(len>=N)                                  /*判断数组长度是否足够插入 x*/
    {   printf("\n 数组存储空间不足, 无法插入! \n");
        return 0;
    }
    else
    {   printf("\n 请输入 x: ");
        scanf("%d", &x);
        for(i=0; i<len; i++)                    /*确定插入位置*/
            if(a[i]>x)  break;
        index=i;
        for(i=len; i>index; i--)                /*从最后一个元素开始后移操作*/
            a[i]=a[i-1];
        a[index]=x;                             /*插入 x*/
        len++;                                  /*数组中有效数据个数加 1*/
        printf("插入%d 后数组：", x);
        for(i=0; i<len; i++)
            printf("%5d", a[i]);
    }
    return 0;
}
```