第5章 数　　组

本章学习目标

- 建立一维数组的概念，以及其在编程中的应用。
- 建立二维数组的概念，以及其在编程中的应用。
- 建立三维及多维数组的概念，以及其在编程中的应用。
- 掌握数组在函数中的应用。
- 熟悉数组在多个常见算法中的使用。
- 能够使用数组实现基本的程序。

5.1　数组产生的背景

数组在信息存储、信息传输、科学计算、信息安全等领域有着广泛的应用。

数组作为一种逻辑上的对类型一致的多个数据存放的方式，在信息安全中占有重要的地位。利用数组，有多种方法可以对其元素进行加密/解密。下面简介几种常见的算法。

（1）置换加密。首先，确立一个密钥，该密钥也是一个数组。按照密钥规定次序来改变明文的元素在其数组中的位置，如此来实现加密数据。这样使得明文数组中各元素的位置被打乱，只有知道置换密码的人才能解密。这种算法的优点是简单易实现，但使用的密钥的安全性对加密效果有很大的影响。

（2）异或加密。首先也是确立一个基于数组形式的密钥。然后将明文数组的每个元素与密钥的对应元素进行异或运算。这种算法也是简单高效，但密钥的选择对加密结果有重要影响。

在信息化极其发达的今天，各种对数据的攻击、窃取、恶意修改等事件日益繁多。对于数组这样一种常见的存储数据的格式，要运用各种加密手段来保护数据的完整性和正确性，由此来保护国家和个人所拥有的数据的安全。因为如今，有时候数组就代表着银行的账号、股票的户头、房产证的序列号，等等。

综上所述，数组作为一种基本的数据存储方式，在当今信息社会的方方面面都有着重要的基石性的作用。

5.2　人以群分、物以类聚

“人以群分、物以类聚”，这是中国的一个成语，也是我们经常所见到的现象。举一个例

子，某军事艺术院校为了迎接建军节，要举办一个盛大的文艺晚会。学校有这么大，学员又这么多，要表演的节目花样繁多，从上百人的合唱到两个人的相声都有，怎么组织学员来表演节目呢？

这里可以把学员按性别来归类。男学员一类，女学员一类。表演一个武装斗争的戏剧，就到男学员这一类中找演员。表演一个优美的歌舞剧，就到女学员这一类中找演员。这就归好类了。假如今晚要排练武装斗争的戏剧，就发通知给男学员。明早要排练歌舞剧，就发通知给女学员。很方便。

假如要准备一个大合唱的节目。在男学员中，根据身高来将他们分为三类：身高在 1.7m 以下的为一类；身高为 1.7～1.8m 的为一类；身高在 1.8m 以上的为一类。然后安排 1.7m 以下的男学员站在合唱团的前面部分，1.7～1.8m 的站在合唱团的中间部分，1.8m 以上的站在合唱团的后面部分。这样，一旦需要彩排或者演出，学员们站位就很简单。每个人都能很快地找到自己的位置。又如，合唱团一般都有声部的划分。类似于刚才的安排，划分起来也容易。例如，前面第一排为高声部，第二排为中声部，等等。假如要练习高声部，那么第一排的出列，然后找个教室，专心练习高声。

所以，“人以群分、物以类聚”，通过合适的分类，我们的工作得到简化，管理变得简单，也大大减少了出错的机会。

案例引入：加解密程序

这里，在 5.1 节的基础上，继续对数组在信息安全中的作用做较深入的探讨。首先，数组可以用来存储待加密的数据。数组的每个存储单元存储待加密数据中的一个。另外，如 5.1 节所述，定义一个密钥数组。

一个例子是对称加密算法。首先使用密钥，这里是存放在一个（一维）数组中。密钥数组对原始数据，也就是待加密数据，进行加密，生成密文。这时生成的密文被命名为 encrypted。这里的密文也可看作一个（一维）数组。然后密文被传输到接收方。在接收方进行解密时，使用与加密时相同的密钥来计算。接收方进行解密后的数组命名为 decrypted。这里加密、传输、解密流程至少需要三个数组的参与，分别是原文数组、密钥数组、密文数组。数组的常用操作如读、写等都有应用。

另一个例子是对数组进行非对称加密。可以把发送者的数据看作一个“流”，或者说是一个（一维）数组，命名为 encrypted；而接收者收到的数据经过解密，也是一个“流”，或者说是一个（一维）数组，命名为 decrypted。加密、解密的工作就是在这两个“流”上面进行的。加密时还需要一个（一维）数组公钥的参与，任何人都可以用公开的这个公钥来加密数据，然后把加密后的数据公开传输给接收方。接收方用自己的私密的私钥来解密收到的数据，也即解密时还需要一个（一维）数组私钥的参与。所以说，整个加密、传输、解密流程共需要至少 4 个数组的参与，分别是原文数组、公钥数组、密文数组、私钥数组。

encrypted 和 decrypted 这两个数组，在加密、解密时分别是计算后的中间结果（encrypted），以及计算后的最终结果（decrypted）。

在加密解密中，二维数组也有应用。AES 算法将明文分成一组一组的，每组长度相同，每次加密一组数据，直到整个明文加密完成。最后将一块一块的密文块拼接起来，形成密文。这里加密后的密文就可以看成一个二维数组（后面会专门讲到）。第一维是所分的各个

组，第二维是每组中的字符内容。

5.3 一维数组

一维数组是最简单的一种数组结构，也是最常用的一种数据结构。一组数据可以存放在一维数组的一连串的单元格里面，然后可以对数组里的数据进行读写操作。指向当前数组成员的指针也可以来回移动，指向不同的数组成员。另外，对C语言中的字符类型char，一维数组有特别的含义：在数组中连续存放的字符类型数据就形成了字符串。

5.3.1 一维数组的声明与初始化

一维数组的定义形式通常在编程语言中表示为

```
数组类型 数组名[常量表达式];
```

在C语言中，具体的例子如：

```
int iScore[5];
```

这里定义了一个名字是“iScore”的数组，数组中共有5个元素。全部元素的数据类型都是整型。

方便起见，在定义数组时，可以一同把数组初始化：

```
int iScore[]= {80,70,95,47,62};
```

下面来看具体的程序例子。

前面学习了变量。例如，有一个同学张三，他的身高是177cm。那对于他的身高，可以定义一个整型变量如下。

```
int iHeightZhangshan;
```

对其赋值如下。

```
iHeightZhangshan = 177;
```

变量iHeightZhangshan代表张三的身高。iHeightZhangshan是这个身高变量的名字。177是这个变量的值。“int”是这个变量的数据类型，这里是整数类型。

上面讨论的是1个同学张三。假如现在有5个同学参加篮球比赛，他们的身高变量及变量的值分别为iHeightZhangshan 177cm，iHeightLisi 180cm，iHeightWangwu 185cm，iHeightZhaoqian 178cm，iHeigthSunli 175cm。

类似于最初对单个的iHeightZhangshan变量的操作，这里可以对这5个同学的身高变量定义并赋值如下。

int iHeigthZhangshan＝177；

int iHeigthLisi＝180；

```
int iHeightWangwu=185;
int iHeightZhaoqian=178;
int iHeigthSunli=175;
```

通过上面5个变量的定义，可以把这支篮球队的身高管理起来。例如，现在有一场比赛，需要查询我方哪个同学个子最高，让他上场。又如，需要计算球队队员的平均身高，以和对手球队的平均身高做比较。

现在看另一个例子。现在有一支长长的登山队伍在羊肠小道上爬华山，如图5-1所示。队伍中共有100名队员。如果把这100名队员的登山队按照上面5名队员的篮球队的方式管理起来，那是不是要为这支长长的登山队伍的每个队员逐个定义变量和赋予他们各自的身高值，共计100次？如果要找出身高最高的队员，是不是要写99次的if-then呢？

这显然是不现实的。为了解决这个问题，数组的作用就显示出来了。

数组是一种数据结构。它包含多个变量，变量的个数可以从0（这时数组为空）到多个（假如对于登山队，这“多个”为100个）。要求这些变量是属于同一种类型。例如，整型或者字符型，或者浮点型，或者双精度型，等等。图5-2是一个数组的存储结构。

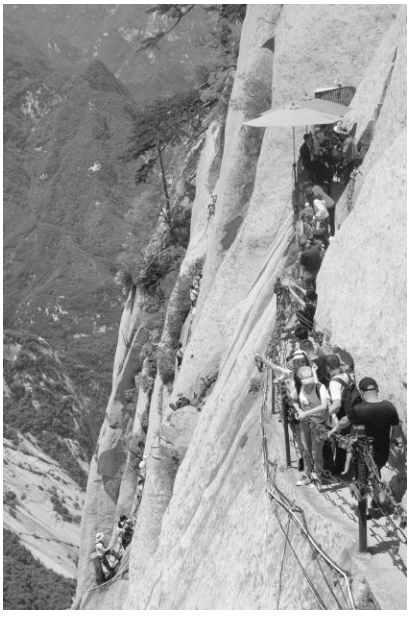

图5-1 爬华山的队伍

图5-2 数组的一个例子

“0,1,2,3,4”是对每个数组元素（成员）的编号（接下来会讲到）。“90,80,70,100,95”是各个元素的值。例如，把某次同学们测验的成绩放在这个数组里，那根据这个存储结构就可以知晓“0”号同学考了“90”分，“1”号同学考了“80”分，“2”号同学考了“70”分，等等。

前述拥有100个成员的华山登山队也就可以用数组定义如下。

```
int iClimbers[100];
```

这里，数组的名字是“iClimbers”。“int”表示该数组的每一个元素都是整数类型。这个数组的元素的最大个数是100个。当然，并不要求数组总是满的，也即合法的元素个数并不

总是100个，可以少于100个。例如，今天30个队员有其他的事要做，不能参与爬山，那今天数组的合法元素只有70个。

假如要查询第67个登山者的身高，对该元素的访问就是：

```
iHeight = iClimbersHeight[66];
printf("第 67 个登山者的身高是 %d\n", iHeight);
```

为什么查询第67个登山者，访问数组元素的方括号里却是66？这里说明一下，在大多数编程语言里，数组的下标是从0开始，也即方括号中的整型数字（下标）是从0开始，而不是从1开始。所以，数组的第1个元素写为iClimbers[0]，第2个元素写为iClimbers[1]，…，第67个元素写为iClimbers[66]，…，第100个元素写为iClimber[99]。

图5-3画出了iClimbers[]数组元素的逻辑地址。

0号元素	1号元素	2号元素	3号元素		98号元素	99号元素
iClimbers[0]	iClimbers[1]	iClimbers[2]	iClimbers[3]	…	iClimbers[98]	iClimbers[99]

图5-3　长数组的存储结构

如果要把第67个登山者的身高打印出来，可以写为

```
printf("第 67 个登山者的身高是%d\n,"iClimbers[66]);
```

上面定义登山队的数组为

```
int iClimbers[100];
```

“[100]”在定义中表示该数组有100个元素（成员）。“int”表示所有数组元素都为整型。其中，第一个元素是iClimbers[0]，最后一个元素是iClimbers[99]。

也可以遍历该数组的所有元素，从数组下标0开始：

```
iClimbers[0],
iClimbers[1],
iClimbers[2],
…
iClimbers[18],
…
iClimbers[99]。
```

类似地，可以有其他数据类型的数组定义：

```
float fTemp[10];            /* 数组有 10 个元素。每个元素都是浮点型数 */
double dMeasurement[40];    /* 数组有 40 个元素。每个元素都是双精度型 */
char cName[20];             /* 数组有 20 个元素。每个元素都是字符型。字符数组和其他数
                               据类型的数组有所不同。后面章节会讲到 */
```

一个基本数据类型的变量，可以在定义这个变量的时候赋初值，也可以在后面的代码中赋值。数组也是类似，可以在定义这个数组的时候赋初值，也可以在后面的代码中赋值。下

面是一个在定义的时候对数组赋初值的例子。

```
int iMonthDays[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

该数组中有 12 个元素，每个元素的值表示对应的月份的天数。
定义的时候，也可以省略方括号中的“12”，如下。

```
int iMonthDays[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

这时候，元素的个数来自于对花括号中的整数个数的计数，计数结果是 12。

5.3.2 数组的元素访问与修改

想要获得第 100 个元素的值，也就是访问第 100 个元素，可以写为

```
int iHeight;
iHeight = iClimbers[99];
```

如果想要修改第 100 个元素的值为 170，就可以写为

```
iClimbers[99] = 170;
```

【例 5.1】 编写程序，用来计算华山登山队队员的平均身高，然后显示出来。

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <time.h>
4   int main(void)
5   {
6       int iClimbers[100] = {0};              //数组所有元素统一赋值为 0
7       int iTotalHeight = 0;
8
9       int i = 0';
10
11      srand((unsigned)time(NULL));           //初始化随机数种子
12      for ( i = 0; i <100; i++)
13      {
14          iClimbers[i] = rand() % 50 + 150;  //产生随机数，身高设置为 150~200cm
15          printf ("%d ",iClimbers[i]);
16          iTotalHeight = iTotalHeight + iClimbers[i];
17      }
18      printf("the average height of all the climbers is %d\n", iTotalHeight/100);
19      return 0;
20  }
```

程序运行结果如下。

```
183 154 193 178 194 162 169 188 195 189 172 171 183 165 158 180 158 169 175 163 184 172
167 161 151 174 191 165 182 165 179 156 185 150 163 173 159 168 159 176 159 183 159 170
```

```
190 197 165 172 189 157 183 185 192 179 186 176 163 170 159 176 160 191 199 150 173 170
153 190 166 193 158 172 176 182 190 160 193 171 157 194 178 161 160 184 196 158 160 157
180 196 160 167 198 163 174 171 180 178 183 181 the average height of all the climbers
is 173
```

案例分析

数组 iClimbers 存储了 100 名登山队员的身高。每个身高均由随机数产生。调用随机数种子 srand((unsigned)time(NULL))一次后，每次调用 rand()即产生一个随机数。每个随机数即每个队员的身高。

iTotalHeight 存储所有队员的身高之和。iTotalHeight/100 即平均身高。

下面是另外一个例子，是关于数组的读和写。

【例 5.2】 编写程序，用于数组的读取和赋值。

```
1   #include <stdio.h>
2   int main(void)
3   {
4       int iRockClimbers[20];
5       int iTotalHeight = 0;
6       int iAverageHeight;
7       int i;
8       printf("Please input heights of all the climbers\n");
9       for ( i = 0; i <20; i++)
10      {
11          scanf("%d", &iRockClimbers[i]);
12          iTotalHeight = iTotalHeight + iRockClimbers[i];
13      }
14      printf("the average height of all the climbers is %d\n", iTotalHeight/20);
15      return 0;
16  }
```

程序运行结果如下。

```
189 177 165 154 167 165 166 167 180 170 175 174 182 173 188 154 155 165 180 165
the average height of all the climbers is 170
```

案例分析

上面的代码与例 5.1 类似。但例 5.1 的身高值的来源是通过随机数产生的。而本例的身高值的来源是通过 scanf()函数从键盘获取用户的输入。

回到例 5.1，需要强调的一点是，在生成访问数组元素的代码这方面，C 语言编译器是不会生成对数组下标（例如，iRockClimbers[7]的下标是 7，是数组的第 8 个元素）进行有效性检查的代码的。并且，有些数组的大小（即包含多少个元素）是程序运行时动态确定的。也就是说，在编译时，编译器无从知晓这类数组的大小，也就无法生成对这类数组的元素进行下标的有效性检查。

有了数组的个数，就有了数组的边界。前面说过，iClimbers[100]数组的元素个数是 100 个，即从 iClimbers[0]到 iClimbers[99]。合法有效的数组下标为[0,…,99]。如果要访

问这个元素 iClimbers[100],这里使用的下标是 100,这是个非法的下标,因为刚才已叙述,下标的最大值只能是 99,所以 iClimbers[100]这个元素是非法元素。对非法元素 iClimbers [100]进行访问而造成的错误,甚至可能导致程序的崩溃,这些都来自代码对数组的访问越界。又如,iClimbers[-1]也是访问数组元素而越界。对程序运行中的数组下标进行有效性检查,这个工作是落在程序员的自己肩上的,由程序员自己来承担。

超越数组边界来进行访问,也是常见的病毒侵入系统的方式之一。

5.3.3 一维数组的常见操作

到目前为止,我们所接触到的数组都是一维数组。它由同一数据类型的元素按照线性顺序排列而成,每个元素都有一个唯一的下标,用于在程序中通过下标访问来读写这些元素。可以想象有一根轴线,一维数组的各个元素就分布在这根轴线上的不同位置。在数学上,可以把一维数组理解成为线性代数里的向量。

如同前面,一个一维数组的定义的例子如下。

```
int iClimbers[100];
```

如同前面,对数组元素访问的一个例子如下。

```
printf("height of an array member No.80: %d\n", iClimbers[80]);
```

对元素进行修改的一个例子如下。

```
iClimbers[80] = 215;
```

这里 iClimbers[80]的值被重新设定为 215。

对数组变量可以通过循环来赋初值,例子如下,每个元素的初值暂定为 170。

```
int i = 0;
for (i = 0; i<= 99; i++)
    iClimbers[i] = 170;
```

这里循环了 100 次,从 0 到 99。但是这段代码有缺陷:这里的 99 是写死在程序里。这种用法缺乏灵活性。相反,可以定义一个整型变量 Count。Count 的值也就是总共循环的次数,或者说是数组成员的个数(这里是 99)。Count 的值可以在 C 程序处由宏定义来设置,或者由用户从键盘输入,或者从一个配置文件中读取,或者由其他程序模块传递而来,等等。这样就灵活得多,99 就不用写死在程序里了。

1. 对一维数组的遍历

【例 5.3】 下面是一个程序片段,对一维数组进行遍历。

```
1   #include <stdio.h>
2   int main(void)
3   {
4       int i = 0;
5       for (i = 0; i <= 99; i++)
```

```
6           printf("iClimbers[%d]: %d\n", i, iClimbers[i]);
7   }
```

案例分析

这里定义了一个循环变量 i。i 循环 100 次，每次循环就把数组 iClimbers[]的对应下标的成员的身高打印出来。

2. 寻找一维数组的最大值和最小值成员

除了遍历数组之外，有时候希望找到数组元素的最大值。

【例 5.4】 编写程序，寻找数组的最大值。

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <time.h>
4
5   int main(void)
6   {
7       int iClimbers[100];
8       int iMax = 0;
9       int i;
10
11      srand((unsigned)time(NULL));
12      for ( i = 0; i <100; i++)
13      {
14          iClimbers[i] = rand() % 50 + 150;
15
16          printf("%d ", iClimbers[i]);
17          if (iClimbers[i] > iMax)
18              iMax = iClimbers[i];
19      }
20      printf("\n the maximum height is %d\n", iMax);
21  }
```

程序运行结果如下。

```
152 176 161 199 171 199 183 156 186 170 173 155 175 159 164 195 177 186 150 186 199 190
188 196 165 190 188 180 160 188 183 167 166 198 153 189 188 185 195 153 183 160 153 182
152 154 153 182 159 178 192 180 160 161 178 165 186 174 188 193 183 191 161 169 171 165
156 178 188 188 195 171 186 196 193 162 193 177 155 187 191 167 156 183 167 164 163 192
177 172 156 183 199 191 180 170 176 164 197 190
The maximum height is 199
```

案例分析

在上述代码中，数组的大小同样是 100。首先通过一个 100 次的循环，每次循环产生一个随机数，将此随机数赋值给对应的数组元素。然后将刚才产生的随机数（身高）与当前的身高的最大值相比较，如果产生的值比最大值大，那么产生的值就被赋值给最大值。

找最小值与找最大值的算法类似，这里不再赘述。

5.4 二维数组

二维数组代表在平面坐标系上相垂直的两个不同方向，一个典型的例子是矩阵。相对于一维数组而言，二维数组在数组的声明、使用等方面都复杂不少。数组涉及的运算一般都是通过两重循环来进行的。例如，首先对行进行循环，然后在每一行内，对列进行循环。在对循环变量的使用、数据访问越界等方面都需要小心。

5.4.1 二维数组的声明与初始化

定义二维数组的一般语法为

```
数据类型 数组名[行数][列数];
```

其中，“行数”和“列数”必须是编译时可以确定的值，如常量或者常量表达式。注意“行”在前面，“列”在后面。

在 C 语言中，具体的例子如：

```
int iParade[4][3];
```

这个整型的二维数组代表一个 4 行、3 列的游行队伍。每个元素的值是队伍中该队员的身高。

下面通过具体的例子来学习二维数组。

华山的羊肠小道上，攀爬的登山队伍可以用一个一维数组来描述。对于一个体育盛会的入场式，一个方阵队伍接着一个方阵队伍。对于一个方阵队伍，可以用一维数组来表示吗？回答是勉强可以。这时，一维数组可以像一条蛇一样弯弯曲曲地把所有的方阵队员串起来，如图 5-4 所示。这就像小汽车的镜子上面的除霜的电热丝，或者像冬天的电热毯里面的电热丝。但是当我们要回答如下问题：方阵第 2 行队员的身高的平均值是多少？第 4 列身高最高的队员是谁？这样的问题时事情就会非常麻烦和容易出错。

图 5-4 汽车后视镜上的电热丝

这时需要引入二维数组。

假设入场方队的人数是 10×20，那么这个方队总共是 200 人。二维数组的声明如下。

```
int iParade[10][20];
```

其中，“iParade”是数组名。“[10]”代表有 10 行，“[20]”代表有 20 列。这里要记清楚，第 1 个下标表示第几行，第 2 个下标表示第几列，前后次序不能颠倒。iParade[]这个数组有 10 行，20 列。它可以看作线性代数里的一个矩阵。

二维数组的初始化有多种方法。一种就是前面在学习一维数组的时候采用的直接赋值的方法——依次对每个成员逐个赋值。但这种方法很烦琐，可读性差。