第5章　数　　组

数组是有序数据的集合，存储同一类型的数据元素，便于对数据进行处理和管理。相对于变量，使用数组可批量定义、管理、操纵多个同类数据对象（数组中的每个数据对象称为数组元素），例如定义数组存储和管理班级内 50 名学生的 C 语言考试成绩；定义数组存储图书馆所有的图书信息。如果要单独为每个学生成绩或每本图书信息定义一个变量进行存储和管理，仅是为变量命名合适的标识符都是件让人头疼的事情，还要记住这些变量名以便在程序中使用，这对程序员来说根本不可能实现，而且毫无效率。本章介绍数组的定义、数组元素的引用及数组初始化等知识，并通过范例讲解和实践任务训练，让读者掌握利用数组编程解决现实问题的基本方法和关键环节，能运用数组实现编程对象的高效管理，提高编程效率。

5.1　知 识 简 介

5.1.1　一维数组

1. 定义

一维数组默认为静态定义，基本方式如下。

类型说明符 数组名[常量表达式];

例如：int a[10];

（1）与变量命名要求一样，数组名必须符合标识符命名规则。

（2）类型说明符确定了每一个数组元素的类型，数组元素具有同类型变量一样的运算规则，占用同样大小的内存空间。

（3）常量表达式可以包含常量或符号常量。常量表达式定义了数组长度，即数组元素的个数。

（4）数组定义后，系统会为数组分配连续的存储空间，数组名代表数组的起始地址（称为**首地址**）。

在上例中，定义了长度为 10 的整型数组 a，系统将为 a 在内存中分配了能存储 10 个整型值的连续空间，a 代表空间的首地址。运行下列代码（部分）。

```
int a[10];
printf("%d,%o\n",sizeof(a),a);      //sizeof 为关键字,sizeof(a)为 a 的空间大小
                                    //%o 八进制输出 a 的值,对应数组首地址
```

运行结果如下：

```
40,30376760

Process returned 0 (0x0)   execution time : 0.235 s
Press any key to continue.
```

注：数组静态定义是在对源程序进行编译时确定数组长度；数组动态定义是在程序运

行过程中确定数组长度。在 C99 标准及其之后的版本中，可以使用变量来定义数组的长度。这被称为变长数组，示例如下所示。

```
int n =5;
int a[n];
```

使用变长数组时，数组的长度在运行时确定，而不是在编译时确定。这意味着变长数组可以根据实际需要动态地分配空间，因此可以节省内存空间。

使用变长数组可能会导致栈溢出、内存泄漏等问题。因此，在使用变长数组时，需要特别注意数组长度的合法性以及内存使用情况，以避免出现潜在的问题。

2. 数组元素的引用

与变量一样，数组也必须先定义，后使用。数组常用的使用方式是对各个数组元素的引用。

数组元素的引用形式如下。

数组名[下标]

对数组元素的引用需包含两个部分信息，一是数组名，代表数组的首地址；二是下标，代表从第一个元素开始，往后移动了几个元素。

数组中第一个元素的下标为 0，表示它从数组首地址开始往后移动 0 个元素。

(1) 下标可以是任何整型常量、整型变量或任何整型表达式。

例如：a[1]=a[2 * 3]+a[3-2];

(2) 数组定义的长度为 LEN，则下标的取值范围为 0～LEN-1。

例如，在 int a[10]中，合法的数组元素对应为 a[0]～a[9]。

注：①C 语言标准并没有要求编译器必须对数组越界进行检查，因此是否进行数组越界检查是由具体的编译器实现决定的。GCC 编译器默认情况下不进行数组越界检查。可以使用编译选项“-Warray-bounds”来开启数组越界检查功能。

② 表达式 a[i]的取值过程是先找到 a+i 的地址，然后再取得该地址中的值；由于 a+i 等于 i+a，因此表达式 a[i]等价于 i[a]。例如 5[a]=15 可实现将数组 a 的第六个元素赋值为 15。但需注意，虽然 5[a]等价于 a[5]，但这种写法可能会让代码变得难以理解，因此不建议在实际编程中使用。

(1) 可以对数组元素赋值，数组元素也可以参与运算，具有与同类型变量一样的运算规则。

(2) 使用数值型数组时，不可以一次引用整个数组，只能逐个引用元素。

3. 初始化

数组的初始化是在定义数组的同时为部分或全部数组元素赋值。数组初始化在编译期进行，可以缩短程序运行的时间，能有效提高运行效率。

(1) 数组初始化时进行整体赋值，在赋值语句中只能为单个元素赋值。例如：

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

(2) 可以只给一部分元素赋值。例如：

```
int a[10]={5,8,7,6};                           //后面没有赋值的元素值默认为 0
```

(3) 对全部数组元素赋值时可以不指定数组长度。例如：

```
int a[10]={0,1,2,3,4,5,6,7,8,9};
int a[ ]={0,1,2,3,4,5,6,7,8,9};                    //二者等价
```

(4) 定义数组时，既不赋初值，也不指定长度是错误的。例如：

```
int a[];                                           //错误
```

5.1.2 二维数组

二维数组定义的一般形式为：

类型说明符 数组名[常量表达式 1][常量表达式 2];

常量表达式 1 常称为二维数组的行数，常量表达式 2 常称为二维数组的列数。

例如：int a[3][4];

a 是 3 行 4 列的二维数组；a 也可以看成是包含 3 个元素的一维数组，每个元素又是含 4 个元素的一维数组。a 中一共包含 3×4＝12 个元素，分别为：

a[0][0]， a[0][1]， a[0][2]， a[0][3]
a[1][0]， a[1][1]， a[1][2]， a[1][3]
a[2][0]， a[2][1]， a[2][2]， a[2][3]

(1) 与一维数组一样，元素下标可以是任何整型常量、整型变量或任何整型表达式。

(2) 二维数组可以在定义的同时整体赋值。例如：

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

(3) 可以把所有数据写在一个花括号内。例如：

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

(4) 可以只对部分元素赋值。例如：

```
int a[3][4]={{1},{5},{9}};                         //其余未赋值的元素默认为 0
int a[3][4]={{1},{5,6}};                           //与下一行初始化结果等价
int a[3][4]={{1,0,0,0},{5,6,0,0},{0,0,0,0}};
```

(5) 对全部数组元素赋值时可以省略第一维长度，第二维不可以省略。例如：

```
a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};    //与下面两行初始化结果等价
a[][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
a[][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

5.1.3 字符数组

1. 一维字符数组的定义及初始化

一维字符数组的定义形式为：

```
char 数组名[常量表达式]
```

例如：

```
char a[10];                                    //字符数组 a 长度为 10
```

(1) 每个元素只能存放一个字符。例如：

```
a[0]='h';a[1]='e';a[2]='l';…
```

(2) 一维字符数组有更多的初始化形式。例如：

```
char a[]={'h','e','l','l','o'};
char a[]="hello";
char a[]={"hello"};
```

注：因为字符串结尾自动加'\0',所以 char a[]="hello";长度为 6,不是 5。

(3) C 语言中没有字符串变量,字符串的输入、存储、处理和输出等必须通过字符数组实现。

2. 字符串的输入

scanf()函数的相关内容如下所示。

(1) 可以用%c 逐个字符输入。例如：

```
char a[10];
for(i=0;i<10;i++)
    scanf("%c",&a[i]);
```

(2) 可以用%s 以字符串的形式输入。例如：

```
char a[10];
scanf("%s",a);
```

注：① a 前不用加 &,因为 a 是数组名,为数组首地址。

② 以%s 输入时,从第一个非空白字符开始,到第一个空白字符终止,例如,输入 Hello world 时,只得到 Hello。

gets()函数的相关内容如下所示。

(1) gets()函数调用方式为：gets(数组名)；

(2) 作用：输入一个字符串,与 scanf();功能一致,但空格和回车都存放在数组中,最后自动加入'\0'。不会出现上面输出不全的情况。

3. 字符串的输出。

printf()函数的相关内容如下所示。

(1) 可以使用%C 逐个字符输出,例如：

```
char a[10];
for(i=0;i<10;i++)
    printf("%c",a[i]);
```

(2) 可以用%s 以字符串的形式输出。例如：

```
char a[10];
```

```
printf("%s",a);
```

puts()函数的相关内容如下所示。

(1) puts()函数调用形式：puts(字符数组名或字符串常量)；

(2) 输出一个字符串，结尾自动换行。

注：gets()函数和 puts()函数需包含头文件"stdio.h"

4. 常用的字符串处理函数(需包含头文件"string.h")

(1) strlen()：测试字符串长度(不包括'\0')。调用方式：

```
strlen(数组名或字符串常量)
```

(2) strcat()：连接两个字符串。调用方式：

```
strcat (字符数组 1,字符数组 2);
       //结果存放在字符数组 1 中
```

(3) strcmp()：比较两个字符串是否相等。调用方式：

```
strcmp (字符串 1,字符串 2);
       //相等时值为 0。字符串 1>字符串 2 时为正数。字符串 1<字符串 2 时为负数。
```

(4) strcpy()：复制字符串。调用方式：

```
strcpy (字符数组 1,字符串 2);
       //字符串 2 的内容复制到字符数组 1 中。字符数组 1 的位置只能是字符数组名。
```

5.2 实践目的

(1) 能深入理解同类型数据对象批量定义的方式，准确分析含有数组元素引用的程序运行结果，判别数组定义、元素引用相关代码的对错，辨识程序优缺点。

(2) 针对多个同类数据对象的表示与使用问题，能通过定义数组，并高效引用数组元素，设计程序流程。

(3) 能够结合选择、循环控制语句，实现对数组元素的操纵，针对程序设计问题中的批量数据处理等问题进行编码实现。

5.3 实践范例

【范例 1】 阅读程序分析结果

要求：编辑下面源程序，分析运行程序结果，然后运行程序，将执行结果与分析结果相对比，完成填空。

```
1   #include<stdio.h>
2   int main()
3   {
4       int n[4]={0},t,j,k=3;
5       for(t=0; t<k; t++)
```

```
6           for(j=0; j<4; j++)
7               n[j]=n[t]+j;
8       for(t=0; t<4; t++)
9           printf("%4d",n[t]);
10      printf("\n");
11      return 0;
12  }
```

分析结果	
运行结果	

范例分析：本范例考查循环嵌套的执行顺序，以及循环执行过程中数组元素的取值和赋值操作。

程序初始化时让数组每个元素为 0，在循环嵌套中为数组元素赋值，最好通过一重循环输出数组元素。在循环嵌套的内层循环中，内层循环变量作为下标的数组元素值会被重新赋值，其值等于内层循环变量值加上当前外层循环变量作为下标的数组元素值。需要注意的是，当内层循环变量和外层循环变量相等时，外层循环变量作为下标的数组元素值会发生变化。数组元素数值变化情况如图 5-1 所示。

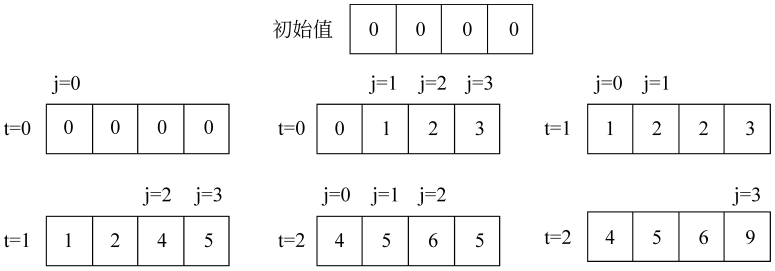
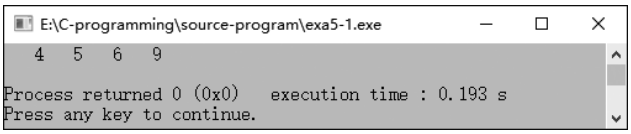


图 5-1　范例 1 数组元素值变化示意图

处理结果：按照以上分析，可完成如下作答。

分析结果	首先，数组 n[4]初始化为每个元素赋值为 0，k 值初始化为 3； 然后，两层循环嵌套(第 5～7 行)中的外层循环变量 t 取值为 0、1、2，内层循环变量 j 为 0 到 3。 当 t＝0 时，a[0]＝0，a[1]到 a[3]在 a[0]基础上分别加上各自的下标，得到 1，2，3； 当 t＝1 时，a[0]＝a[t]＝1，a[1]＝a[1]＋1＝2，然后 a[2]和 a[3]等价于 a[1]＋2＝4，a[1]＋3＝5； 当 t＝2 时，a[0]＝a[2]＝4，a[1]＝a[2]＋1＝5，a[2]＝a[2]＋2＝6，a[3]＝a[2]＋3＝6＋3＝9。最后，输出数组各元素的值，分别为 4，5，6，9
运行结果	

【范例 2】 补充程序

下面代码功能为：筛选法求 1～200 间的素数。请将代码补充完整。不要增行或删行、

改动程序结构。

```
1   #include<stdio.h>
2   int main()
3   {
4       int a[201],i,j;
5       for(i=1; i<=200; i++)
6           a[i] =i;                                    //为数组赋初值
7       a[1] =0;                                        //先去掉 a[1]
8       for(i=2; i<15; i++)
9       {
10          if(!a[i])
11          ____________;
12          for(j=i+1; j<=200; j++)
13              if(_________________)
14                  a[j] =0;
15      }
16      int n;
17      for(i=2,n=0; i<=200; i++)
18      {
19          if(________)
20          {
21              printf("%5d",a[i]);
22              n++;
23          }
24          if(n==10)
25          {
26              printf("\n");
27              n =0;                                   //一次完成之后初始化
28          }
29      }
30      return 0;
31  }
```

范例分析：筛选法是一种由希腊数学家埃拉托斯特尼提出的简单检定素数的算法，全称为埃拉托斯特尼筛法。其算法过程如下所示。

(1) 去掉 1；

(2) 用下一个未被去掉的数 p 除 p 后面各数，把 p 的倍数去掉；

(3) 检查 p 是否小于根号 n 的整数部分，如果是，则返回(2)继续执行，否则结束；

(4) 剩下的就是素数。

利用筛选法找到 1～30 中的素数过程如图 5-2 所示。

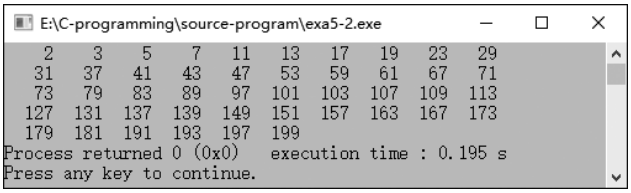
基于算法描述，代码中首先定义数组存储 1～200 的数值(第 5、6 行)，然后去掉 1(第 7 行)。接下来要实现的功能是，对于 i 从 2 到 14(小于根号 200，第 8 行)，若 a[i]已被去掉则结束本次循环(第 10、11 行，本算法中，去掉某数字就是将对应数组的元素赋值为 0)；否则

图 5-2　筛选法求 1～30 素数过程示意图

转向内层循环(第 12～14 行),去掉是 a[i]倍数的元素(第 13、14 行),然后转到 a[i+1]再重复执行前面的操作。

在第一个空(第 11 行),不执行后续代码而跳转到 i+1 的操作,可用关键字 continue;第二个空(第 13 行)为让 a[j]赋值为 0 的条件,其核心关键是 a[j]能整除 a[i];第三个空(第 19 行)需填写输出 a[i]的条件,应该是 a[i]未被去掉,即 a[i]不为 0。

处理结果：根据上述分析,可补充填空为 continue、a[j]! =0 && (a[j]%a[i] == 0)、a[i]! =0。补充代码后,程序运行结果如下。



注：上面的代码存在可以优化的地方,即当 a[i]不为 0 时,只需将 200 以内的 a[i]的倍数直接赋值为 0 即可。代码为 for(j=2,k=200/i;j<=k;j++) a[i*j]=0;。

【范例 3】 调试程序

要求：用户输入字符串 b,与程序制定的字符串 a 比较大小,输出比较结果。分析已给出的语句,要求判断调试运行该程序是否正确,若有错,写出错在何处,并填写正确的运行结果。不要增行或删行、改动程序结构。

```
1   #include <stdio.h>
2   #include <string.h>
3   int main()
4   {
5       char a[10]="beijing",b[10];
6       int i;
7       gets(b);
```

```
8       printf("a=%s,b=%s\n",a,b);
9       for(i=0;a[i]!='\0'&&b[i];i++)
10          if(a[i]>b[i])
11          {
12              printf("a:%s >b:%s\n",a,b);
13              break;
14          }
15          else if(a[i]<b[i])
16          {
17              printf("a:%s <b:%s\n",a,b);
18              break;
19          }
20      if(a[i])
21          printf("a:%s >b:%s\n",a,b);
22      else if(b[i])
23          printf("a:%s <b:%s\n",a,b);
24      else
25          printf("a:%s ==b:%s\n",a,b);
26      return 0;
27  }
```

若有错，指出错误并修改	错误行号：	应改为：
	错误行号：	应改为：
	错误行号：	应改为：
调试正确后的运行结果	输出结果：	

范例分析：本范例考查字符数组/字符串操作的常见功能实现，C 语言 string.h 中的库函数读者可以尝试自己编码实现其功能。上述程序仿照 strcmp()函数实现两个字符串的大小比较，比较方法如下。

(1) 比较字符串中的第一个字符。如果它们相同，且都存在未比较的字符，则继续比较下一个字符，直到找到不同的字符。

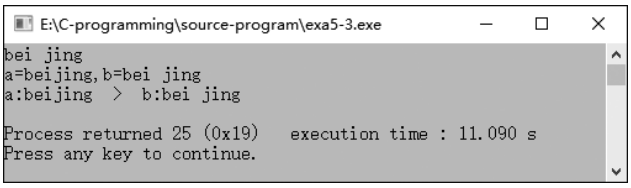
(2) 如果找到不同的字符，则比较字符 ASCII 码的大小，ASCII 码较大的字符串大，ASCII 码较小的字符串小。比较结束，输出结果。

(3) 若其中一个字符串已经结束，而另一个字符串还有未比较的字符，则另一个串较大。

(4) 如果两个串同时结束且前面的字符按下标对应相同，则认为它们相等。

通过分析，在 8～18 行的循环中，若两个字符不相等时，输出结果后应结束程序，而不仅是跳出循环(跳出循环后会执行后面的 if 语句，可能导致输出错误)。

处理结果：根据分析，可完成填空如下。

若有错，指出错误并修改	错误行号：12	应改为：return 0；
	错误行号：17	应改为：return 0；
	错误行号：	应改为：
调试正确后的运行结果	输出结果：（输入 bei jing） 	

【范例 4】 编写程序

范例描述：某班有 56 名学生，年龄在 18 岁到 24 岁之间，整型数组 age[70]按学号从小到大的顺序存储每名同学的年龄。请编程统计每个年龄的人数并输出。

范例分析：对于上述问题，需要进行以下步骤来求解。

（1）**数据准备**：确定输入输出。输入可包括班级总人数（56 人）、年龄范围（18 岁到 24 岁）以及每个学生的年龄。输出是每个年龄的人数统计。

（2）**数据结构的选择**：考虑年龄是有限的整数范围（18 岁到 24 岁），可使用一维整型数组来表示每个年龄的人数统计。同时，需要一维整型数组来存储每个学生的年龄数据。

（3）**初始化计数数组**：在开始统计之前，需要将用于统计每个年龄人数的数组中的元素初始化为 0。

（4）**输入数据并统计**：循环输入每个学生的年龄数据，同时进行统计。假设输入的第 i 个学生的年龄 age[i]的值为 20，则统计时需要将 count[20－18]的值加 1，即 count[20－18]＋＋。由于不能确定用户输入的 age[i]的值的大小，因此 count[20－18]＋＋中的 20 需要用 age[i]代替，统计语句应改为 count[age[i]－18]＋＋；

（5）**输出结果**：完成统计后，需要遍历计数数组，输出每个年龄的人数统计结果。

范例代码：根据上述分析，可编写代码如下所示。

```
#include <stdio.h>
int main()
{
    int age[70];                          // 声明整型数组用于存储学生年龄
    int n =56;                            // 学生总数
    int minAge =18;                       // 最小年龄
    int maxAge =24;                       // 最大年龄
    int ageRange =maxAge -minAge +1;      // 年龄范围
    int count[ageRange];                  // 声明计数数组
    for (int i =0; i <ageRange; i++)      // 初始化计数数组
        count[i] =0;
    // 输入学生年龄并统计每个年龄的人数
    printf("请输入 %d 名学生的年龄(%d 岁到%d 岁): \n", n, minAge, maxAge);
    for (int i =0; i <n; i++)
    {
```