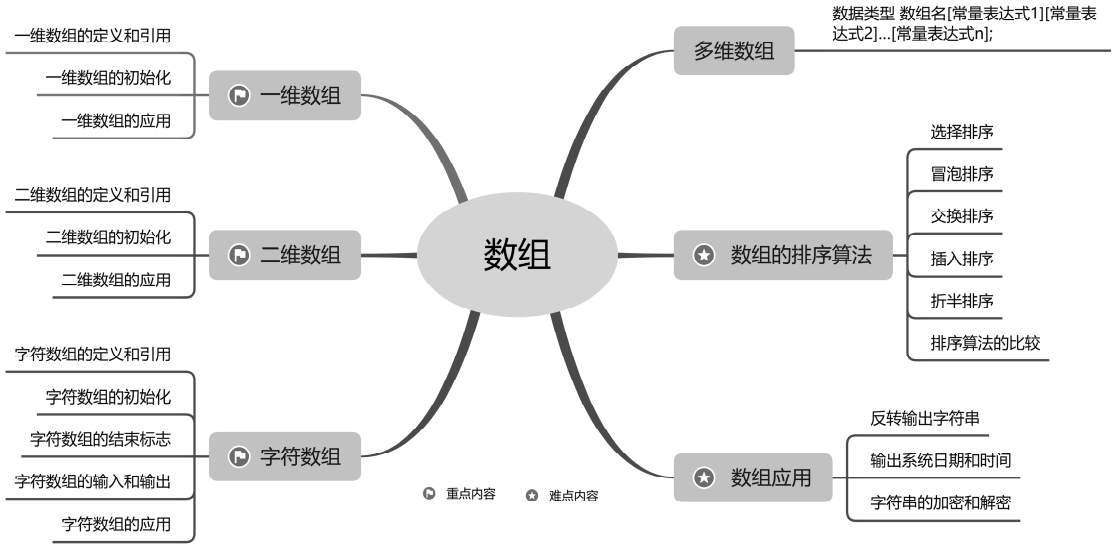
第 8 章

数组

编写程序过程中，有时会用到大量相同类型的数据。如果每个变量都需要单独定义，编程过程将会变得极其烦琐。使用数组可以很好地解决这个问题。本章致力于使读者掌握一维数组和二维数组的应用，能利用所学知识解决一些实际问题；掌握字符数组的使用及其相关操作；并能通过数组掌握常见的排序算法。

本章的知识架构及重难点如下：



8.1 一 维 数 组

数组是一个由若干相同类型变量组成的集合，引用这些变量可以使用同一个名字。数组均由连续的存储单元组成，最低地址对应于数组的第一个元素，最高地址对应于数组的最后一个元素。数组可以是一维数组，也可以是多维数组。

8.1.1 一维数组的定义和引用

一维数组实际上是一组相同类型数据的线性集合，其示意图如图 8.1 所示。

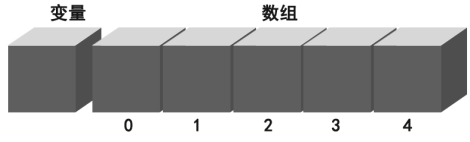


图 8.1 一维数组示意图

1．一维数组的定义

一维数组的定义形式如下：

```
类型说明符 数组标识符[常量表达式];
```

☑ 类型说明符：表示数组中元素的类型。
☑ 数组标识符：表示该数组变量的名称，命名规则与变量名一致。
☑ 常量表达式：定义了数组中存放的数据元素个数，即数组长度。

例如，定义一个包含 5 个整型元素的数组，代码如下：

```
int iArray[5];
```

代码中的 int 为数组元素的类型，iArray 为数组变量名，括号中的 5 表示数组中包含 5 个元素。

注意

int [3]={1,3,4}，这样定义是错误的。在定义数组时必须要有数组标识符。

2．一维数组的引用

数组定义后，可以引用其中的数组元素，引用方式为“数组标识符[下标]”。数组下标可以是整型常量或整型表达式。

例如，引用数组 iArray 中的第 3 个变量，格式为 iArray[2]。其中，iArray 是数组变量名，2 为数组下标。有的读者会问：为什么引用第 3 个数组元素使用的是下标 2 呢？这是因为数组下标是从 0 开始的，iArray[0]表示第一个元素，iArray[1]表示第 2 个数组元素，iArray[2]表示第 3 个数组元素。

注意

在数组 iArray[5]中，只能使用 iArray[0]、iArray[1]、iArray[2]、iArray[3]、iArray[4]，而不能使用 iArray[5]。若使用 iArray[5]，会出现下标越界错误。

【例 8.1】使用数组保存手机号（**实例位置：资源包\TM\sl\08\01**）

在本实例中，使用数组保存用户输入的手机号，并输出显示。

```
#include<stdio.h>
int main()
{
    int iArray[11], index;                          /*定义数组及变量为基本整型*/
    printf("请输入手机号:\n");
    for (index= 0; index< 11; index++)              /*逐个输入手机号码，保存为数组元素*/
    {
        scanf("%d",&iArray[index]);
    }
    printf("手机号是:\n");
    for (index = 0; index< 11; index++)             /*循环输出数组中的元素*/
    {
        printf("%d", iArray[index]);
    }
    printf("\n");
    return 0;
}
```

（1）定义 index 表示循环控制变量，定义数组 iArray[11]用来保存 11 位手机号码。

（2）通过一个 for 循环，依次输入 11 位手机号码（用空格隔开），存储到各数组元素对应的地址中。这里，iArray[index]就是对数组元素的引用，“&”为取地址符。

（3）循环输出各数组元素，得到完整的手机号码。

运行程序，显示效果如图 8.2 所示。

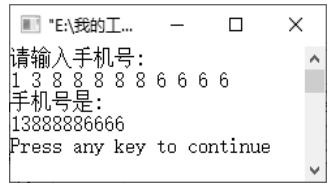


图 8.2　使用数组保存手机号

8.1.2　一维数组的初始化

一维数组的初始化，可以用以下 3 种方法实现。

（1）定义数组时直接对数组元素赋初值（数组元素值放在一对大括号中）。例如：

```
int i,iArray[6]={1,2,3,4,5,6};
```

定义和初始化之后，iArray[0]=1，iArray[1]=2，iArray[2]=3，iArray[3]=4，iArray[4]=5，iArray[5]=6。

（2）如果只给一部分数组元素赋值，则未赋值的元素默认为被赋值 0。例如：

```
int iArray[6]={0,1,2};
```

数组 iArray 包含 6 个元素，但初始化时只给出了 3 个值，结果是数组前 3 个元素得到赋值，后 3 个元素被默认赋值为 0。

（3）当对全部数组元素都赋初值时，可以不指定数组长度。例如：

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

上述代码的大括号中有 4 个元素，因此系统会默认该数组变量的长度为 4。

【例 8.2】计算篮球平均成绩（**实例位置：资源包\TM\sl\08\02**）

记分员记录了某球员在 10 场篮球比赛中的成绩，求该球员的平均成绩。

```
#include<stdio.h>
int main()
{
    int grade[10]={12,5,21,15,32,10,25,14,30,20};      /*定义数组，存放球员的 10 场比赛成绩*/
    int total=0;                                       /*定义变量 total，表示总成绩*/
    int i;
    float avg;                                         /*定义变量 avg，表示平均成绩*/
    for(i=0;i<10;i++)                                  /*循环累计总成绩*/
    {
        total+=grade[i];
    }
    avg=((float)total/10);                             /*计算平均成绩，计算前将 total 强制转化为实数*/
    printf("篮球比赛的平均成绩是：%f\n",avg);
    return 0;
}
```

程序中首先定义一个数组 grade，初始化数值为 10 场球赛的成绩；定义整型变量 total，用来保存总成绩；定义浮点型变量 avg，用来保存平均成绩。接着通过一个 for 循环，使 10 场球赛成绩累加，相加结果赋给 total。最后计算平均成绩，赋给 avg，并用 printf 函数输出结果。

运行程序，显示效果如图 8.3 所示。

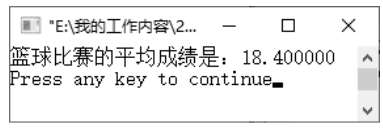


图 8.3　初始化一维数组

8.1.3　一维数组的应用

一个班级中往往有很多学生，使用数组来保存这些学生的姓名、编号等，管理起来非常方便。

【例 8.3】输出插队之后的编号（实例位置：资源包\TM\sl\08\03）

体育课上，老师按身材高矮给 20 名同学编号。刚编完号，一位男生姗姗来迟，老师比对身高后将他排在 8 号位置，并重新排列后面的同学。使用数组，输出男生插队后有变化的学生编号。代码如下：

```
#include<stdio.h>
int main()
{
	/*定义数组及变量为基本整型*/
	int iArray[20]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}, index;
	int iArray1[21]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21}, index1;
	printf("体育老师按身材高矮排队编号，老师排好编号是:\n"); /*提示信息*/
	for (index = 0; index< 20; index++)                    /*循环输出插队前所有学生的编号*/
	{
		printf("%d ", iArray[index]);
	}
	printf("\n");
	printf("重新排列插队男生及后面的同学，他们的编号是:\n");    /*男生插入后，从 8 号开始重新编号*/
	for (index1 = 7; index1< 21; index1++)                 /*从下标 7 开始，循环输出后续编号*/
	{
		printf("%d ", iArray1[index1]);
	}
	printf("\n");
	return 0;
}
```

可以看出，要想从第 8 名同学开始编号，就应该从下标 7 开始。运行程序，效果如图 8.4 所示。

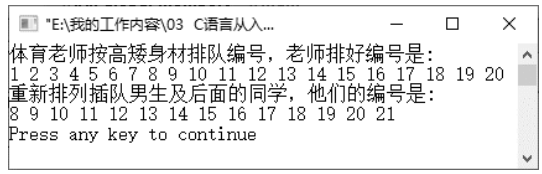


图 8.4　重新输出编号

编程训练（答案位置：资源包\TM\sl\08\编程训练\）

训练 1：保存语、数、外成绩　编写程序，输入语文、数学、英语 3 门学科的成绩，使用数组保存并输出。运行效果如下：

```
请输入语文、数学、英语的成绩:
89 95 96
语文、数学、英语的成绩分别如下:
89 95 96
```

训练 2：双十一购物　模拟输出小莉的双十一购物车清单。运行效果如下：

```
购物车清单:

====生活用品类:====
纸抽,纸巾,收纳箱,水杯,垃圾袋,剪刀,挂钩,拖鞋,小闹钟

====化妆品类:====
保湿套装,气垫 cc,隔离霜,防晒霜,眉粉,眼影色盘,睫毛膏

====运动类商品:====
运动服,球鞋,护腕,护膝,护掌,排球,瑜伽垫,瑜伽球
```

```
====保健类商品:====
蛋白粉,口服液,眼部按摩仪,血压计,脚底按摩器
```

8.2 二 维 数 组

一维数组的下标只有一个，如果下标有两个呢？C 语言中，将下标有两个的数组称为二维数组。二维数组包含行、列两个维度，如 iArray[m][n]就表示一个包含 m 行 n 列，共计 m×n 个元素的数组。同样，通过行下标和列下标，可以快速地定位二维数组中的任意一个元素。

例如，图 8.5 所示的房间号索引图就可以看作是一个 4 行 7 列的二维数组。要想找到 4104 房间，需要先定位行，找到 4 楼，再定位列，找到第 4 列，就可以准确找到 4104 房间。

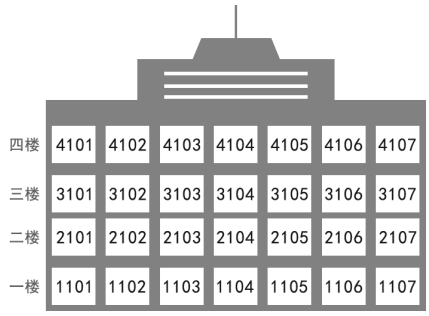


图 8.5 房间号索引示意图

8.2.1 二维数组的定义和引用

1. 二维数组的定义

二维数组可以看作是特殊的一维数组，其各元素仍然是一个数组。其定义形式如下：

```
数据类型 数组名[常量表达式 1][常量表达式 2];
```

其中，“常量表达式 1”定义了二维数组的行数，“常量表达式 2”定义了二维数组的列数。

不管是行下标还是列下标，其索引都是从 0 开始的。因此，一个 m 行 n 列的二维数组 array[m][n]，其行下标取值范围为 0～m−1，列下标取值范围为 0～n−1，最大下标元素是 array[m−1][n−1]。

例如，定义一个 3 行 4 列的整型数组，代码如下：

```
int array[3][4];
```

二维数组 array[3][4]共包含 12 个数组元素（3×4），分别为 array[0][0]、array[0][1]、array[0][2]、array[0][3]、array[1][0]、array[1][1]、array[1][2]、array[1][3]、array[2][0]、array[2][1]、array[2][2]和 array[2][3]。

可见，C 语言中数组是按行排列的，数组 array[3][4]在内存中先按行顺次存放，依次为 array[0]行、array[1]行和 array[2]行，每行有 4 个元素，也依次存放。

2. 二维数组的引用

二维数组元素的引用形式为“数组名[下标][下标]”。例如，array[1][2]表示对 array 数组的第 2 行第 3 个元素进行引用。

和一维数组一样，二维数组也要注意下标越界的问题。例如：

```
int array[2][4];          /*定义一个 2 行 4 列的二维数组*/
…                         /*对数组元素进行赋值*/
array[2][4]=9;            /*错误，下标越界！ */
```

8.2.2　二维数组的初始化

二维数组赋初值，有以下 4 种情况。

（1）将所有数据写在一个大括号内，按照数组元素排列顺序对元素赋值。例如：

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

如果大括号内的数据少于数组元素的个数，则系统会默认后面未被赋值的元素值为 0。

（2）为所有元素赋初值时，可以省略行下标，但不能省略列下标。例如：

```
int array[][3] = {1,2,3,4,5,6};
```

系统会根据数据的个数进行分配，一共有 6 个数据，而数组分为 3 列，因此数组有 2 行。

（3）分行给数组元素赋值。例如：

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

在分行赋值时，可以只对部分元素赋值。例如，下面的代码中，a[0][0]=1，a[0][1]=2，a[0][2]=0，a[1][0]=4，a[1][1]=5，a[1][2]=0。

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

（4）直接对数组元素赋值。例如：

```
int a[2][3];
a[0][0] = 1;
a[0][1] = 2;
```

【例 8.4】魔方阵数据（**实例位置：资源包\TM\sl\08\04**）

一个 3×3 的网格，将 1～9 的数字放入方格中。通过键盘为二维数组元素赋值，并显示二维数组。

```
#include<stdio.h>
int main()
{
    int a[3][3];                          /*定义一个 3 行 3 列的二维数组*/
    int i,j;                              /*定义两个循环控制变量*/
    for(i=0;i<3;i++)                      /*先行后列，依次输入数据，为数组元素赋值*/
    {
        for(j=0;j<3;j++)
        {
            printf("a[%d][%d]=",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    printf("输出二维数组:\n");
    for(i=0;i<3;i++)                      /*先行后列，循环输出所有数组元素的值*/
    {
        for(j=0;j<3;j++)
        {
            printf("%d\t",a[i][j]);       /*使用制表符“\t”控制间距，使排列更整齐*/
        }
        printf("\n");                     /*使数组元素分行显示*/
    }
    return 0;
}
```

（1）程序先依次输入相应的数组元素值，然后将这个 3 行 3 列的数组输出。

（2）给数组赋值时，使用了 for 循环嵌套，第一层循环行，第二层循环列，用 printf 函数输出元素名，用 scanf 函数输入数据，赋给对应的数组元素。

（3）输出数组时，也使用了 for 循环嵌套，第一层循环行，第二层循环列，用 printf 函数逐行输出数组元素。为了使输出的数据更整齐，使用制表符“\t”来控制间距。

运行程序，显示效果如图 8.6 所示。

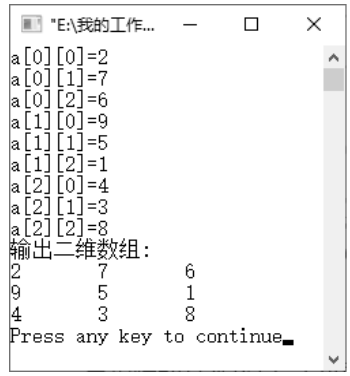


图 8.6　使用二维数组保存数据

8.2.3　二维数组的应用

【例 8.5】将矩阵行列对换（**实例位置：资源包\TM\sl\08\05**）

本实例中，把二维数组中各行的元素换成列元素，各列的元素换成行元素，生成一个新的二维数组。代码如下：

```
#include<stdio.h>
int main()
{
    int a[2][3],b[3][2];                    /*定义两个二维数组*/
    int i,j;                                /*定义两个循环控制变量*/

    for(i=0;i<2;i++)                        /*先行后列，输入数据，为数组 a 的各元素赋值*/
    {
        for(j=0;j<3;j++)
        {
            printf("a[%d][%d]=",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    printf("输出二维数组:\n");
    for(i=0;i<2;i++)                        /*先行后列，循环输出数组 a 中的元素*/
    {
        for(j=0;j<3;j++)
        {
            printf("%d\t",a[i][j]);
        }
        printf("\n");                       /*使数组元素分行显示*/
    }
    for(i=0;i<2;i++)                        /*将数组 a 转置后存入数组 b 中*/
    {
        for(j=0;j<3;j++)
        {
            b[j][i] = a[i][j];              /*通过行列互换，使得 b 为 a 的转置数组*/
        }
    }
    printf("输出转换后的二维数组:\n");
    for(i=0;i<3;i++)                        /*先行后列，循环输出转置数组 b 中的元素*/
    {
        for(j=0;j<2;j++)
        {
```

```
                printf("%d\t",b[i][j]);
            }
            printf("\n");                    /*使数组元素分行显示*/
        }
        return 0;
}
```

（1）利用双层 for 循环为 2 行 3 列的数组赋值，然后分行显示其元素。

（2）通过一个双层 for 循环，将二维数组 a 中的元素赋值到转置后的二维数组 b 中。

（3）通过循环控制，将转置后的二维数组 b 中的元素输出。

运行程序，显示效果如图 8.7 所示。

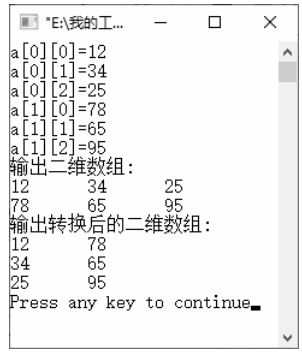


图 8.7　矩阵转置运行图

编程训练（答案位置：资源包\TM\sl\08\编程训练\）

训练 3：打印数组对角线上的字符　有如下二维数组：

```
char ccArray[5][5] = {
        {'a','b','c','d','e'},
        {'b','a','8','d','d'},
        {'c','d','a','e','c'},
        {'d','j','f','a','b'},
        {'e','d','a','f','a'},
};
```

编写程序，打印数组对角线（左上至右下）上的字符。输出结果如下：

```
对角线上的字符是:a
对角线上的字符是:a
对角线上的字符是:a
对角线上的字符是:a
对角线上的字符是:a
```

训练 4：求平均成绩　用二维数组求表 8.1 中各科的平均成绩。

表 8.1　成绩表

	宋小美	张大宝	高心心	彭果	邓丽
数学	93	87	90	76	70
语文	90	76	60	80	81
英语	70	88	72	77	96

运行结果如下：

```
请输入成绩:
array[0][0]=93
array[0][1]=87
array[0][2]=90
array[0][3]=76
array[0][4]=70
array[1][0]=90
array[1][1]=76
array[1][2]=60
array[1][3]=80
array[1][4]=81
array[2][0]=70
array[2][1]=88
array[2][2]=72
array[2][3]=77
array[2][4]=96
```

```
数学的平均成绩是（取整数）:83
语文的平均成绩是（取整数）:77
英语的平均成绩是（取整数）:80
```

8.3 字 符 数 组

数组中的元素类型为字符型时，称为字符数组。字符数组中的每个元素可以存放一个字符。字符数组的定义和引用方法与其他数组类型相似。

8.3.1 字符数组的定义和引用

C 语言中没有专门的字符串变量，没有 string 类型，通常使用字符数组来存放字符串。字符数组实际上是一系列的字符集合，不严谨地说就相当于字符串。例如，定义一个字符数组 iArray[6]，按照如图 8.8 所示的初始化形式，将会在控制台上输出“MingRi”。

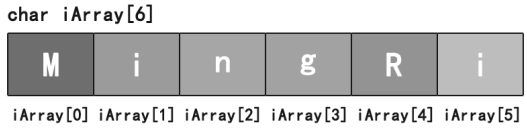


图 8.8 定义一个字符数组 iArray[6]

字符数组的定义形式如下：

```
char 数组标识符[常量表达式]
```

例如，下述代码定义了一个字符数组 cArray，该数组中包含 5 个字符型的变量元素。

```
char cArray[5];
```

字符数组的引用也采用下标的形式。例如，cArray[2]表示对 cArray 数组中第 3 个字符进行引用。也可以在引用数组元素的同时进行赋值。

```
cArray[0]='h';
cArray[1]='e';
cArray[2]='l';
cArray[3]='l';
cArray[4]='o';
```

8.3.2 字符数组的初始化

在对字符数组进行初始化操作时，有以下几种方法。

（1）逐个字符赋给数组中的元素。这是最容易理解的初始化字符数组的方式。

例如，下述代码定义了一个包含 5 个元素的字符数组，在初始化大括号中为每个数组元素赋值。

```
char cArray[5]={'H','e','l','l','o'};
```

（2）定义字符数组的同时进行初始化，此时可以省略数组长度。

如果初值个数与预定的数组长度相同，在定义时可以省略数组长度，系统会自动根据初值个数来确定数组长度。例如，上面初始化字符数组的代码可以写成：

```
char cArray[]={'H','e','l','l','o'};
```

代码中定义的 cArray[]没有给出数组大小，但系统会自动根据初值的个数确定数组长度为 5。

（3）利用字符串给字符数组赋初值。

字符数组可用来存放字符串。因此，也可以用字符串的方式对字符数组进行初始化赋值。例如：

```
char cArray[]={"Hello"};
```

或者将“{}”去掉，写成：

```
char cArray[]="Hello";
```

【例 8.6】打印停车场标志（**实例位置：资源包\TM\sl\08\06**）

本实例将利用字符数组输出“Park”。定义一个字符数组，通过初始化操作保存一串字符，然后通过循环引用将每一个数组元素输出。

```
#include<stdio.h>
int main()
{
    char cArray[5]={'P','a','r','k'};          /*定义字符数组并初始化*/
    int i;                                     /*定义循环控制变量*/
    for(i=0;i<5;i++)                           /*循环输出字符数组元素*/
    {
        printf("%c",cArray[i]);
    }
    printf("\n");                              /*输出换行*/
    return 0;                                  /*程序结束*/
}
```

初始化字符数组时要注意，每个元素都要使用一对单引号（''）括起来。在 for 循环中，因为输出的类型是字符型，所以 printf 函数中使用的是“%c”格式符。通过循环变量 i，cArray[i]实现了对数组中不同元素的引用。

运行程序，显示效果如图 8.9 所示。

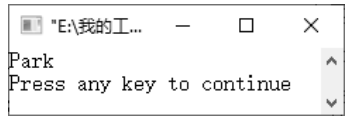


图 8.9　使用字符数组输出一个字符串

8.3.3　字符数组的结束标志

C 语言中，使用字符数组保存字符串时，系统会自动为其添加“\0”作为结束符。也就是说，用字符串方式赋值比用字符逐个赋值要多占一个字节，多占的这个字节用于存放字符串结束标志“\0”。

例如，使用下述代码可以初始化一个字符数组：

```
char cArray[]="Hello";
```

上面的字符数组 cArray 在内存中的实际存放情况如图 8.10 所示。

H	e	l	l	o	\0

图 8.10　cArray 在内存中的实际存放情况

“\0”是由 C 编译系统自动加上的。因此上面的赋值语句等价于：

```
char cArray[]={'H','e','l','l','o','\0'};
```

字符数组并不要求最后一个字符为“\0”，甚至可以不包含“\0”。因此下面的写法也是合法的：

```
char cArray[5]={'H','e','l','l','o'};
```

是否加“\0”，应根据需要来决定。由于系统对字符串常量会自动加一个“\0”，因此，为了使处理方法一致，且便于测定字符串的实际长度，在字符数组中也常常人为地加上一个“\0”。例如：

```
char cArray[6]={'H','e','l','l','o','\0'};
```

注意

如果一个字符数组需要先后存放多个不同长度的字符串，则应使数组长度大于最长的那个字符串的长度。

8.3.4 字符数组的输入和输出

字符数组的输入和输出可以使用两种格式字符：“%c”和“%s”。

使用格式字符“%c”，可逐个输入与输出数组中的字符，类似于一维数组。例如，下述代码逐个输出字符数组 cArray 中的元素。

```
for(i=0;i<5;i++)                                    /*进行循环*/
{
    printf("%c",cArray[i]);                         /*逐个字符输出数组元素*/
}
```

使用格式字符“%s”，可将整个字符串输入或输出。例如，下述代码输出字符串"GoodDay!"。

```
char cArray[]="GoodDay!";                           /*初始化字符数组*/
printf("%s",cArray);                                /*输出字符串*/
```

使用格式字符“%s”输出字符串时，需要注意以下几种情况。

☑ 输出字符中不包括结束符“\0”。

☑ printf 函数中的输出项是字符数组名 cArray，而不是数组中的元素名 cArray[0]等。

☑ 即使数组长度大于字符串实际长度，也只会输出到“\0”为止。

☑ 如果一个字符数组中包含多个“\0”结束字符，则在遇到第一个“\0”时输出就结束。

【例 8.7】输出一条心灵鸡汤（**实例位置：资源包\TM\sl\08\07**）

输出名言“Education is the door to freedom”（教育是通向自由之门）。对定义的字符数组初始化，然后采用两种方式输出字符数组中保存的数据，先逐个字符输出数组元素，再直接将字符串整体输出。

```
#include<stdio.h>
int main()
{
    int iIndex;                                     /*循环控制变量*/
    char cArray[33]="Education is the door to freedom";   /*定义字符数组，用于保存字符串*/
    char cArray2[100]="教育是通向自由之门";

    for(iIndex=0;iIndex<33;iIndex++)
    {
        printf("%c",cArray[iIndex]);                /*逐个字符输出数组元素*/
    }
    printf("\n%s\n",cArray2);                       /*将字符串整体输出*/
    return 0;
}
```

在代码中，对数组元素逐个字符输出时，使用的是循环方式。直接输出字符串时，使用的是格式

字符“%s”。要注意，直接输出字符串时不能使用格式字符“%c”。

运行程序，显示效果如图 8.11 所示。

注意

C 语言中只存在字符类型，不存在字符串类型。字符类型只能存放单个字符，如果要存放字符串，就需要使用字符数组。当然，也可以使用字符型指针存放字符串（详见第 10 章）。

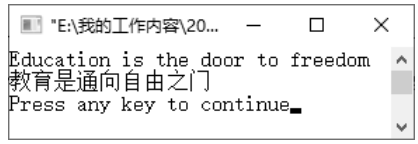


图 8.11　使用两种方式输出字符串

8.3.5　字符数组的应用

【例 8.8】求解字符串长度（**实例位置：资源包\TM\sl\08\08**）

随机输入一个字符串，求解该字符串的长度。

```
#include<stdio.h>
int main()
{
    int iIndex;                                  /*循环控制变量*/
    int length=0;                                /*定义变量 length，保存字符串长度*/
    char cArray[80];                             /*定义字符数组，保存用户输入的字符串*/
    printf("请输入字符串：\n");
    gets(cArray);                                /*用 gets 函数输入字符串*/
    for(iIndex=0;cArray[iIndex]!='\0';iIndex++)  /*循环字符数组，直到遇到“\0”结束*/
    {
        length++;                                /*每遍历一个字符，字符串长度加 1*/
    }
    printf("字符串长度是：%d\n",length);          /*输出字符串长度*/
    return 0;
}
```

使用 gets 函数将输入的字符串保存在 cArray 字符数组中。使用 for 循环判断当前数组元素是否为结束符“\0”，如果不是，则字符串长度加 1；如果是，则退出循环。

运行程序，显示效果如图 8.12 所示。

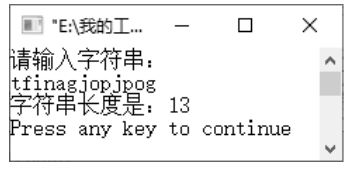


图 8.12　计算字符串长度

编程训练（答案位置：资源包\TM\sl\08\编程训练\）

训练 5：统计单词个数　用户输入一行字符，然后统计其中有多少个单词。要求每个单词之间用空格分隔开，且最后的字符不能为空格。输出结果如下：

```
请输入字符串：
I Love China
一共有 3 个单词
```

训练 6：确定男女主角　利用字符数组保存男女主角的名字，使用 for 循环遍历字符数组，通过两种形式分别输出男女主角的名字。输出结果如下：

```
这部电影的男主角分别是：
雨石,玉轩,团子
这部电影的女主角分别是：
小点,紫轩,若美
```

8.4 多 维 数 组

多维数组的声明和二维数组相同，只是下标更多。其一般形式如下：

```
数据类型 数组名[常量表达式 1][常量表达式 2]...[常量表达式 n];
```

例如，下面的代码中定义了一个三维数组 iArray1 和一个四维数组 iArray2。

```
int iArray1[3][4][5];
int iArray2[4][5][7][8];
```

由于数组元素的位置可以通过偏移量计算出来，因此对于三维数组 a[m][n][p]来说，元素 a[i][j][k]所在的地址是从 a[0][0][0]算起到 i*n*p+j*p+k 个单位的位置。

8.5 数组的排序算法

数组是一组有序数据的集合。这里，“有序”指的是数组元素在内存中的存放方式是有序的，其引用方式也有规律可循，而不是说数组元素在数组中是按照数值大小有序排列的。那么，有没有可能让数组元素按照数值大小有序排列呢？当然可以，下面就一起来学习下数组的各种常用排序算法。

8.5.1 选择排序

选择排序的原理如下：每次在待排序数组中查找最大或最小的数组元素，将其与前面没有进行过排序的数组元素互换。这里，由大到小排序应查找最大值，由小到大排序则应查找最小值。

下面以数字 9、6、15、4、2 为例，利用选择法使其从小到大排序，每次交换后的数字顺序如图 8.13 所示。

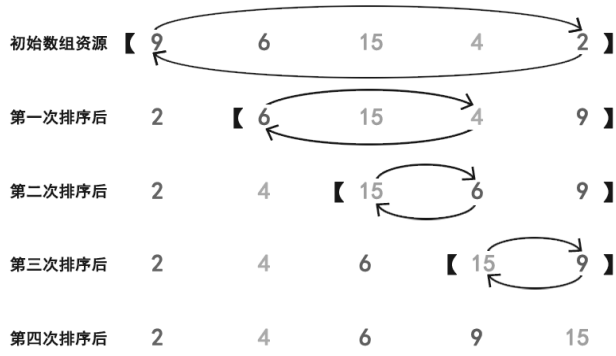


图 8.13 选择排序示意图

可以发现，第一次排序过程中，将第一个数字 9 和整个数组中最小的数字 2 进行了位置互换；第二次排序过程中，将第二个数字 6 和剩下数字中最小的数字 4 进行了位置互换；依此类推，每次都将下一个数字和剩余数字中最小的数字进行位置互换，直到将一组数字按从小到大排序。

【例 8.9】选择法从小到大排序（**实例位置：资源包\TM\sl\08\09**）

本实例中，声明了一个整型数组和两个整型变量。其中，整型数组用于存储用户输入的 10 个数值，而整型变量用于存储数值最小的数组元素和该元素的位置。通过双层循环进行选择法排序，最后将排好序的数组元素输出。

```
#include <stdio.h>
int main()
{
	int i,j;
	int a[10];                          /*定义数组，存储用户输入的 10 个数*/
	int iTemp;                          /*定义变量，表示最小的数组元素*/
	int iPos;                           /*定义变量，表示元素位置*/
	printf("为数组元素赋值：\n");
	for(i=0;i<10;i++)                   /*输入 10 个数，为数组元素赋值*/
	{
		printf("a[%d]=",i);
		scanf("%d", &a[i]);
	}

	/*使用选择法对数组元素从小到大排序*/
	for(i=0;i<9;i++)                    /*设置外层循环下标为 0～8，表示前 9 个数组元素*/
	{
		iTemp = a[i];                   /*假设当前元素为最小值*/
		iPos = i;                       /*记录最小元素位置*/
		for(j=i+1;j<10;j++)             /*设置内层循环下标为 i+1～9，表示剩下的未排序数组元素部分*/
		{
			if(a[j]<iTemp)              /*如果后续的元素中有数比前面设定的最小值还小*/
			{
				iTemp = a[j];           /*重新设定最小值*/
				iPos = j;               /*修正最小元素位置*/
			}
		}
		a[iPos] = a[i];                 /*此两行代码用于将最小的数组元素和当前排序次数对应的数组元素互换*/
		a[i] = iTemp;
	}
	printf("排序结果如下：\n");
	for(i=0;i<10;i++)                   /*输出数组*/
	{
		printf("%d\t",a[i]);            /*用制表位分隔数据*/
		if(i == 4)                      /*如果是第 5 个元素*/
			printf("\n");               /*输出换行*/
	}
	printf("\n");
	return 0;                           /*程序结束*/
}
```

（1）声明一个整型数组，并通过键盘输入为数组元素赋值。

（2）设置一个嵌套循环，第一层循环为前 9 个数组元素，并在每次循环时将对应当前次数的数组元素设置为最小值（如果当前是第 3 次循环，那么将数组中第 3 个元素，也就是下标为 2 的元素设置为当前的最小值）；在第二层循环中，循环比较该元素之后的各个数组元素，并将每次比较结果中较小的数设置为最小值，在第二层循环结束时，将最小值与开始时设置为最小值的数组元素进行互换。当所有循环都完成以后，就将数组元素按照从小到大的顺序重新排列了。

（3）循环输出数组中的元素，并在输出 5 个元素以后换行，在下一行输出后面的 5 个元素。

运行程序，显示效果如图 8.14 所示。

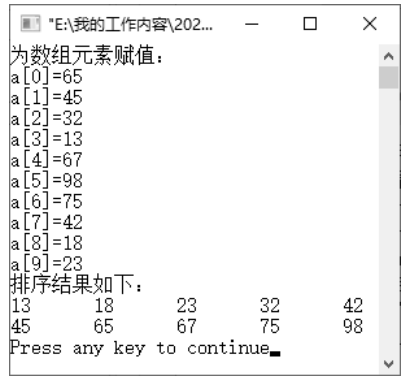


图 8.14　选择排序结果

8.5.2 冒泡排序

冒泡排序的原理如下：依次比较数组中相邻两个数组元素的值，每次都将较小的数排在较大的数前面，可实现数组元素从小到大排序；每次都将较大的数排在较小的数前面，可实现数组元素从大到小排序。

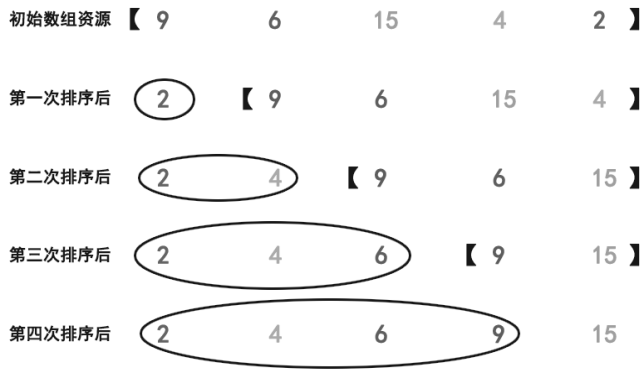


图 8.15 冒泡排序示意图

仍以数字 9、6、15、4、2 为例，对这几个数字进行冒泡排序，使其从小到大排列。每次排序后的顺序如图 8.15 所示。

可以发现，在第一次冒泡排序过程中，将最小的数字 2 移动到第一的位置（“冒泡”上浮），并将其他数字依次向后移动；在第二次冒泡排序过程中，从第二个数字开始的剩余数字中选择最小的数字 4，并将其移动到第二的位置，剩余数字依次向后移动；依此类推，每次都将剩余数字中最小的数字移动到当前剩余数字的最前方，直到将一组数字按从小到大排序为止。

【例 8.10】冒泡法从小到大排序（**实例位置：资源包\TM\sl\08\10**）

声明一个整型数组和一个整型变量，整型数组用于存储用户输入的数字，整型变量作为两个元素交换时的中间变量，通过双层循环进行冒泡排序，最后将排好序的数组输出。

```
#include<stdio.h>
int main()
{
    int i,j;
    int a[10];
    int iTemp;
    printf("为数组元素赋值：\n");
    for(i=0;i<10;i++)                    /*输入 10 个数，为数组元素赋值*/
    {
        printf("a[%d]=",i);
        scanf("%d", &a[i]);
    }

    /*采用冒泡法使数组元素从小到大排序*/
    for(i=1;i<10;i++)                    /*外层循环元素的下标为 1～9，表示后 9 个数组元素*/
    {
        for(j=9;j>=i;j--)                /*内层循环元素的下标为 9～i，表示从最后一个元素开始向前循环*/
        {
            if(a[j]<a[j-1])              /*如果前一个数比后一个数大*/
            {
                /*交换两个数组元素的值，使小数前移，如同冒泡*/
                iTemp = a[j-1];
                a[j-1] = a[j];
                a[j] = iTemp;
            }
        }
    }
```

```
    printf("排序结果如下：\n");
    for(i=0;i<10;i++)                    /*输出数组*/
    {
        printf("%d\t",a[i]);             /*用制表位分隔数据*/
        if(i == 4)                       /*如果是第 5 个元素*/
            printf("\n");                /*输出换行*/
    }
    printf("\n");
    return 0;                            /*程序结束*/
}
```

（1）声明一个整型数组，并通过键盘输入为数组元素赋值。

（2）设置一个嵌套循环，第一层循环为后 9 个数组元素。在第二层循环中，从最后一个数组元素开始向前循环，直到前面第一个没有进行排序的数组元素。循环比较这些数组元素，如果在比较中后一个数组元素的值小于前一个数组元素的值，则将两个数组元素的值进行互换。当所有循环都完成以后，就将数组元素按照从小到大的顺序重新排列了。

（3）循环输出数组中的元素，并在输出 5 个元素以后进行换行，在下一行输出后面的 5 个元素。

运行程序，显示效果如图 8.16 所示。

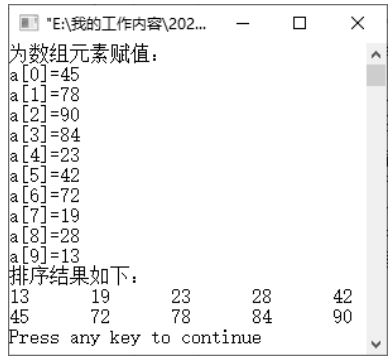


图 8.16　冒泡排序结果

8.5.3　交换排序

交换排序的原理如下：将每一位数与其后的所有数一一比较，如果发现符合条件的数据，则交换数据。首先，用第一个数依次与其后的所有数进行比较，如果存在比其值大（小）的数，则交换这两个数，继续向后比较其他数直至最后一个数；然后再使用第二个数与其后面的数进行比较，如果存在比其值大（小）的数，则交换这两个数；继续向后比较其他数，直至最后一个数比较完成。

下面以数字 9、6、15、4、2 为例，采用交换法实现数字按从小到大进行排序，每次排序的结果如图 8.17 所示。

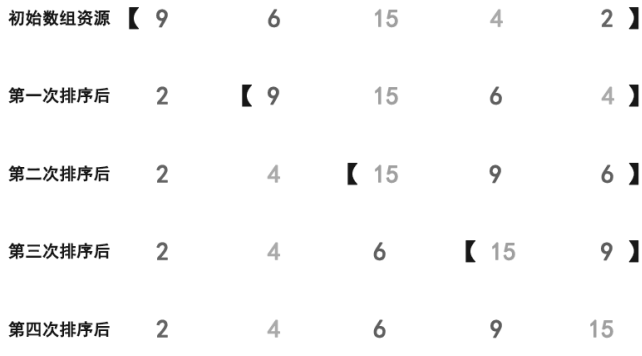


图 8.17　交换排序示意图

可以发现，在第一次排序过程中将第一个数字 9 与后边的数依次进行比较。首先比较 9 和 6，9 大于 6，交换两个数的位置，然后数字 6 成为第一个数字；用 6 和 15 进行比较，6 小于 15，保持原来的位置；然后用 6 和 4 进行比较，6 大于 4，交换两个数字的位置；再用当前数字 4 与最后的数字 2 进行比较，4 大于 2，交换两个数字的位置，从而得到图 8.16 中第一次的排序结果。然后使用相同的方法，从当前第二个数字 9 开始，继续和后面的数字依次比较，如果遇到比当前数字小的数字则交换位置。依此类推，直到将一组数字全部按从小到大排序为止。

【例 8.11】交换法从小到大排序（**实例位置：资源包\TM\sl\08\11**）

声明一个整型数组和一个整型变量，其中整型数组用于存储用户输入的数字，整型变量则作为两个元素交换时的中间变量，然后通过双层循环进行交换法排序，最后将排好序的数组输出。

```
#include<stdio.h>
int main()
{
    int i,j;
    int a[10];
    int iTemp;
    printf("为数组元素赋值：\n");
    for(i=0;i<10;i++)                    /*输入 10 个数，为数组元素赋值*/
    {
        printf("a[%d]=",i);
        scanf("%d", &a[i]);
    }

    /*采用交换法使数组元素从小到大排序*/
    for(i=0;i<9;i++)                     /*外层循环元素下标为 0～8，表示前 9 个数组元素*/
    {
        for(j=i+1;j<10;j++)              /*内层循环元素下标为 i+1～9，表示后面的待比较数组元素*/
        {
            if(a[j] < a[i])              /*如果后一个数比前一个数小*/
            {
                iTemp = a[i];            /*交换两个数组元素的值，使小数前移*/
                a[i] = a[j];
                a[j] = iTemp;
            }
        }
    }
    printf("排序结果如下：\n");
    for(i=0;i<10;i++)                    /*输出数组*/
    {
        printf("%d\t",a[i]);             /*用制表位分隔数据*/
        if(i == 4)                       /*如果是第 5 个元素*/
            printf("\n");                /*输出换行*/
    }
    printf("\n");
    return 0;                            /*程序结束*/
}
```

（1）声明一个整型数组，并通过键盘输入为数组元素赋值。

（2）设置一个嵌套循环，第一层循环为前 9 个数组元素，然后在第二层循环中，将第一个数组元素与后面的数组元素依次进行比较，如果后面的数组元素值小于当前数组元素值，则交换两个元素值，然后使用交换后的第一个数组元素继续与后面的数组元素进行比较，直到本次循环结束。将最小的数组元素值交换到第一个数组元素的位置，然后从第二个数组元素开始，继续与后面的数组元素进行比较。依此类推，直到循环结束，就将数组元素按照从小到大的顺序重新排列了。

（3）循环输出数组中的元素，并在输出 5 个元素以后进行换行，在下一行输出后面的 5 个元素。

运行程序，显示效果如图 8.18 所示。

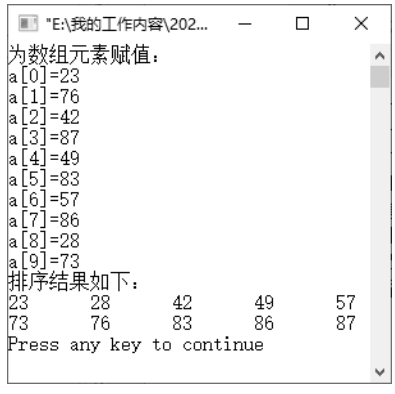


图 8.18　交换排序结果

8.5.4　插入排序

插入排序较为复杂，其基本原理是：抽出一个数据，在前面的数据中寻找相应的位置插入，然后继续下一个数据，直到完成排序。

下面以数字 9、6、15、4、2 为例，使用插入法使其从小到大排序。排序过程如图 8.19 所示。

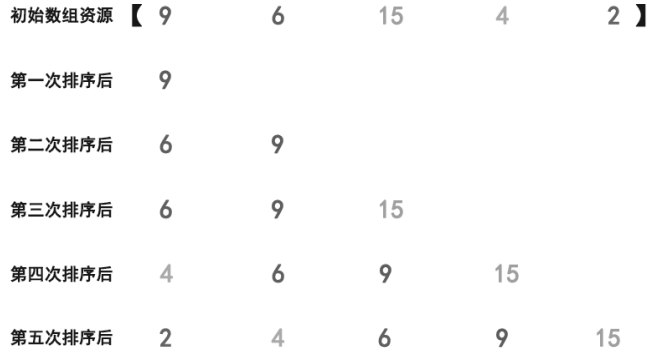


图 8.19　插入排序示意图

可以发现，在第一次排序过程中将第一个数字 9 取出来，并放置在第一个位置。然后取出第二个数字 6，并与第一个数字进行比较，如果第二个数字小于第一个数字，则将第二个数字排在第一个数字之前，否则排在第一个数字之后。然后取出第 3 个数字，先与排在最后面的数字进行比较，如果当前数字比较大，则继续排在最后；如果当前数字比较小，还要与之前的数字进行比较。如果当前数字比前面的数字小，则将其排在比它小的数字和比它大的数字之间；如果没有比当前数字小的数字，则将当前数字排在最前方。依此类推，不断取出未进行排序的数字，与排好序的数字进行比较，并插入相应的位置，直到将一组数字全部按从小到大实现排序为止。

【例 8.12】插入法从小到大排序（**实例位置：资源包\TM\sl\08\12**）

声明一个整型数组和两个整型变量，其中整型数组用于存储用户输入的数字，一个整型变量作为两个元素交换时的中间变量，一个用于记录数组元素的位置，然后通过双层循环进行交换法排序，最后将排好序的数组输出。

```
#include<stdio.h>
int main()
{
    int i;
    int a[10];
    int iTemp;
    int iPos;
    printf("为数组元素赋值：\n");
    for(i=0;i<10;i++)                          /*输入 10 个数，为数组元素赋值*/
    {
        printf("a[%d]=",i);
        scanf("%d", &a[i]);
    }

    /*采用插入法使数组元素从小到大排序*/
    for(i=1;i<10;i++)                          /*外层循环元素下标为 1～9，表示后 9 个数组元素*/
    {
        iTemp = a[i];                          /*设置插入值*/
        iPos = i-1;
        while((iPos>=0) && (iTemp<a[iPos]))    /*内层循环，寻找插入值的位置*/
        {
            a[iPos+1] = a[iPos];               /*插入数值*/
            iPos--;
```

```
            }
            a[iPos+1] = iTemp;
        }
        printf("排序结果如下：\n");
        /*输出数组*/
        for(i=0;i<10;i++)
        {
            printf("%d\t",a[i]);                    /*用制表位分隔数据*/
            if(i == 4)                              /*如果是第 5 个元素*/
                printf("\n");                       /*输出换行*/
        }
        printf("\n");
        return 0;                                   /*程序结束*/
}
```

（1）声明一个整型数组，并通过键盘输入为数组元素赋值。

（2）设置一个嵌套循环，第一层循环为后 9 个数组元素，将第二个元素赋值给中间变量，并记录前一个数组元素的下标位置。在第二层循环中，首先要判断是否符合循环的条件，允许循环的条件是记录的下标位置必须大于等于第一个数组元素的下标位置，并且中间变量的值小于之前设置下标位置的数组元素，如果满足循环条件，则将设置下标位置的数组元素赋值给当前的数组元素，然后将记录的数组元素下标位置向前移动一位，继续进行循环判断。内层循环结束以后，将中间变量中保存的数值赋值给当前记录的下标位置之后的数组元素，继续进行外层循环，将数组中下一个数组元素赋值给中间变量，再通过内层循环进行排序，依此类推，直到循环结束，就将数组元素按照从小到大的顺序重新排列了。

（3）循环输出数组中的元素，并在输出 5 个元素以后进行换行，在下一行输出后面的 5 个元素。

运行程序，显示效果如图 8.20 所示。

```
选择"E:\我的工作内容\...
为数组元素赋值：
a[0]=12
a[1]=76
a[2]=43
a[3]=25
a[4]=79
a[5]=97
a[6]=54
a[7]=32
a[8]=21
a[9]=11
排序结果如下：
11      12      21      25      32
43      54      76      79      97
Press any key to continue
```

图 8.20　插入排序结果

8.5.5　折半排序

折半排序又称为快速排序，其基本原理为：选择一个中间值（在程序中使用数组中间元素的值），然后把比中间值小的元素放在左边，比中间值大的元素放在右边（具体的实现是从两边查找，找到一对后进行交换），然后再对左右两边分别递归使用折半法排序过程。

说明

折半法又叫二分法，在 n 个数中排序，只需要排 log(n)次。

下面以数字 9、6、15、4、2 为例，对这几个数字使用折半法从小到大排序。每次排序后的顺序如图 8.21 所示。

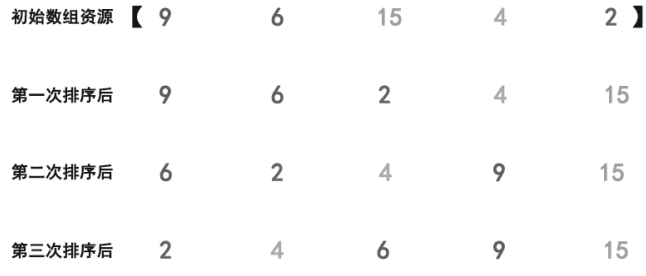


图 8.21　折半排序示意图

可以发现，在第一次排序过程中，首先获

取数组中间元素的值 15，从左右两侧分别取出数组元素与中间值进行比较。如果左侧取出的值比中间值小，则取下一个数组元素与中间值进行比较；如果左侧取出的值比中间值大，则交换两个互相比较的数组元素值。右侧的比较正好与左侧相反，当右侧取出的值比中间值大时，取前一个数组元素的值与中间值进行比较；如果右侧取出的值比中间值小，则交换两个互相比较的数组元素值。当中间值两侧的数据都比较一遍以后，左侧以第一个元素为起点，以中间值的元素为终点，用上面的方法继续进行比较；右侧则以中间值的元素为起点，以数组最后一个元素为终点，用上述的方法继续进行比较。当比较完成以后，继续以折半的方式进行比较，直到将一组数字全部按从小到大的顺序排序为止。

注意

折半法排序中会用到函数递归调用（即函数调用自身），这部分内容将在第 9 章中介绍，这里了解即可。读者也可以参考后面的内容提前进行学习。

【例 8.13】折半法从小到大排序（**实例位置：资源包\TM\sl\08\13**）

在本实例中声明了一个整型数组，用于存储用户输入的数字，再定义一个函数，用于对数组元素进行排序，最后将排好序的数组输出。

```
#include<stdio.h>
void CelerityRun(int left, int right, int array[]);          /*声明函数*/

int main()
{
    int i;
    int a[10];
    printf("为数组元素赋值：\n");
    for(i=0;i<10;i++)                                        /*输入 10 个数，为数组元素赋值*/
    {
        printf("a[%d]=",i);
        scanf("%d", &a[i]);
    }
    CelerityRun(0,9,a);                                      /*调用折半排序函数*/
    printf("排序结果如下：\n");
    for(i=0;i<10;i++)                                        /*输出数组*/
    {
        printf("%d\t",a[i]);                                 /*用制表位分隔数据*/
        if(i == 4)                                           /*如果是第 5 个元素*/
            printf("\n");                                    /*输出换行*/
    }
    printf("\n");
    return 0;                                                /*程序结束*/
}

void CelerityRun(int left, int right, int array[])           /*定义折半排序函数*/
{
    int i,j;
    int middle,iTemp;
    i = left;
    j = right;
    middle = array[(left+right)/2];                          /*求中间值*/
    do
    {
        while((array[i]<middle) && (i<right))                /*从左侧查找小于中间值的数*/
            i++;
```

```
            while((array[j]>middle) && (j>left))        /*从右侧查找大于中间值的数*/
                j--;
            if(i<=j)                                    /*如果找到一对值*/
            {
                iTemp = array[i];                       /*交换两个值*/
                array[i] = array[j];
                array[j] = iTemp;
                i++;
                j--;
            }
        }while(i<=j);                                   /*如果两边的下标交错，就停止（完成一次）*/
        if(left<j)                                      /*递归左半边*/
            CelerityRun(left,j,array);
        if(right>i)                                     /*递归右半边*/
            CelerityRun(i,right,array);
}
```

（1）声明一个整型数组，并通过键盘输入为数组元素赋值。

（2）定义折半排序函数 CelerityRun，用于对数组元素进行排序，3 个参数分别表示递归调用时数组最开始元素的下标、最后元素的下标以及要排序的数组。声明两个整型变量，作为控制排序算法循环的条件，分别将两个参数赋值给变量 i 和 j，i 表示左侧下标，j 表示右侧下标。首先使用 do…while 语句设计外层循环，条件为 i≤j，表示如果两边的下标交错，就停止循环。内层的两个循环分别用来比较中间值两侧的数组元素，当左侧的数值小于中间值时，取下一个元素与中间值进行比较，否则退出第一个内层循环；当右侧的数值大于中间值时，取前一个元素与中间值进行比较，否则退出第二个内层循环。然后判断 i 的值是否小于等于 j，如果是，则交换以 i 和 j 为下标的两个元素值，继续进行外层循环。当外层循环结束以后，以数组第一个元素到以 j 为下标的元素为参数递归调用该函数，同时，以 i 为下标的数组元素到数组最后一个参数也作为参数递归调用该函数。依此类推，直到将数组元素按照从小到大的顺序重新排列为止。

（3）循环输出数组中的元素，并在输出 5 个元素以后进行换行，在下一行输出后面的 5 个元素。

运行程序，显示效果如图 8.22 所示。

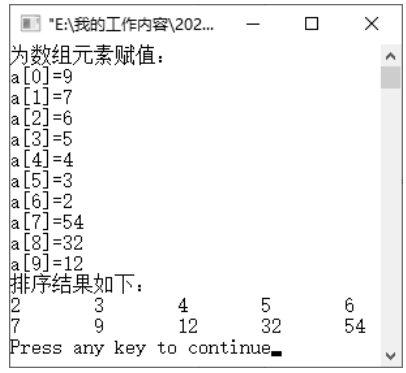


图 8.22　折半排序结果

8.5.6　排序算法的比较

前面介绍了 5 种排序算法，在具体应用时应该怎么选择呢？下面对 5 种排序算法的擅长方向进行总结。

- ☑ 选择排序：简单、容易实现，适用于数量较小的排序。排序过程中需要进行 n(n−1)/2 次比较，互相交换 n−1 次。
- ☑ 冒泡排序：相对稳定的排序方法，当待排序列有序时，效果比较好。最好的情况是正序，只要比较一次即可；最坏的情况是逆序，需要比较 n^2 次。
- ☑ 交换排序：和冒泡排序类似，正序时最快，逆序时最慢，排列有序数据时效果最好。
- ☑ 插入排序：要经过 n−1 次插入过程，如果数据恰好位于插入序列的最后端，则不需要移动数据，可节省时间。因此，若原始数据基本有序，具有较快的运算速度。

☑ 折半排序：n 越大，速度越快。n 很小时，比其他排序算法都慢。折半排序是不稳定的，当有相同关键字的记录时，排序后的结果可能会颠倒次序。

总之，插入排序、冒泡排序、交换排序的速度较慢，但当参加排序的序列局部或整体有序时，这几种排序能达到较快的速度；在这种情况下，折半排序反而会显得速度慢了。当 n 较小，对稳定性不作要求时，宜选用选择排序法；对稳定性有要求时，宜选用插入排序或冒泡排序法。

编程训练（答案位置：资源包\TM\sl\08\编程训练\）

训练 7：电视剧收视率排名　利用交换排序法将以下电视剧收视率从高到低排序。

《Give up,hold on to me》	收视率：1.4%
《The private dishes of the husbands》	收视率：1.343%
《My father-in-law will do martiaiarts》	收视率：0.92%
《North Canton still believe in love》	收视率：0.862%
《Impossible task》	收视率：0.553%
《Sparrow》	收视率：0.411%
《East of dream Avenue》	收视率：0.164%
《The prodigal son of the new frontier town》	收视率：0.259%
《Distant distance》	收视率：0.394%
《Music legend》	收视率：0.562%

程序运行结果如下：

```
1.4000   1.3930   0.9200   0.8620   0.5620
0.5530   0.4110   0.3940   0.2590   0.1640
```

训练 8：公司股票排名　分析 10 家公司的股票数据，判断哪家更值得投资。声明一个整型数组和一个整型变量，其中整型数组用于存储用户输入的数字，而整型变量则作为两个元素交换时的中间变量，然后通过双层循环进行冒泡排序，最后将排好序的数组输出。输出结果如下：

```
各公司股票数据如下:
a[0]=546789
a[1]=543879
a[2]=234780
a[3]=357698
a[4]=345275
a[5]=875422
a[6]=875606
a[7]=567548
a[8]=459078
a[9]=438997
234780   345275   357698   438997   459078
543879   546789   567548   875422   875606
```

8.6　数 组 应 用

本节将运用数组知识，通过 3 个实例讲解实际开发中常遇到的一些问题。

8.6.1 反转输出字符串

【例 8.14】反转输出字符串（**实例位置：资源包\TM\sl\08\14**）

以字符串 mrsoft 为例，其反转的结果为 tfosrm。其算法实现过程如图 8.23 所示。定义两个字符数组，一个表示源字符串，另一个表示反转后的目标字符串。在源字符串中从第一个字符开始遍历，将第一个字符赋给目标字符串的最后一个字符，将第二个字符赋给目标字符串的倒数第二个字符，依此类推，就实现了字符串的反转。

```
#include<stdio.h>
int main()
{
    int i;
    char String[7] = {"mrsoft"};                    /*定义源字符串*/
    char Reverse[7] = {0};                          /*定义反转字符串*/
    int size;
    size = sizeof(String);                          /*计算源字符串长度*/
    for(i=0;i<6;i++)                                /*循环读取字符*/
    {
        Reverse[size-i-2] = String[i];              /*源字符串倒序存入反转字符串*/
    }
    printf("输出源字符串：%s\n",String);             /*输出源字符串*/
    printf("输出目标字符串：%s\n",Reverse);          /*输出目标字符串*/
    return 0;                                       /*程序结束*/
}
```

运行程序，显示效果如图 8.24 所示。

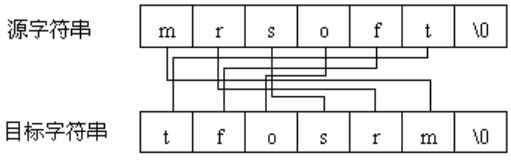


图 8.23 字符串反转示意图

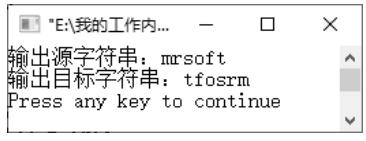


图 8.24 反转输出字符串

8.6.2 输出系统日期和时间

【例 8.15】输出系统日期和时间（**实例位置：资源包\TM\sl\08\15**）

设计一个程序，当用户输入 0 时显示帮助信息，输入 1 时显示系统日期，输入 2 时显示系统时间，输入 3 时退出系统。

要实现上述功能，需要解决两个问题：一是要不断地保持程序运行，等待用户输入命令，防止 main 函数结束；二是要获取系统日期和时间。第一个问题可使用一个无限循环语句来实现，在循环语句中等待用户输入，如果用户输入的是 3，则终止循环，结束应用程序。第二个问题需要使用时间函数 time 和 localtime 来实现。

在 main 函数中将各个控制命令保存在数组中，然后使用 while 语句设计一个无限循环。在循环中让用户输入命令，并判断用户输入的命令是否和数组中存储的命令相同。如果相同，执行相应的语句。

```
#include<stdio.h>                                   /*包含头文件 stdio.h*/
#include<time.h>                                    /*包含头文件 time.h*/
```

```
int main()
{
    int command[4] = {0,1,2,3};                    /*定义一个数组*/
    int num;
    struct tm *sysTime;
    printf("如需帮助可输入数字 0！\n");              /*输出提示信息*/
    printf("请输入命令符：\n");

    while (1)                                      /*通过 while 循环，使系统始终等待用户输入*/
    {
        scanf("%d", &num);                         /*用户输入数字*/
        /*判断用于输入的字符*/
        if(command[0] == num)                      /*如果用户输入数字 0*/
        {
            /*输出帮助信息*/
            printf("输入数字 1 显示系统日期，输入数字 2 显示系统时间，输入数字 3 退出系统！\n");
        }
        else if(command[1] == num)                 /*如果用户输入数字 1*/
        {
            time_t nowTime;
            time(&nowTime);                        /*获取系统日期*/
            sysTime= localtime(&nowTime);          /*转换为本地日期*/
            printf("系统日期：%d-%d-%d \n",1900 + sysTime->tm_year,sysTime->tm_mon + 1,sysTime-> tm_mday);
                                                   /*输出信息*/
        }
        else if(command[2] == num)                 /*如果用户输入数字 2*/
        {
            time_t nowTime;
            time(&nowTime);                        /*获取系统时间*/
            sysTime = localtime(&nowTime);         /*转换为本地时间*/
            printf("系统时间：%d:%d:%d \n",sysTime->tm_hour ,sysTime->tm_min ,sysTime-> tm_sec);
                                                   /*输出信息*/
        }
        else if(command[3] == num)                 /*如果用户输入数字 3*/
        {
            return 0;                              /*退出系统*/
        }
        printf("请输入命令符：\n");                 /*输出提示信息*/
    }
    return 0;                                      /*程序结束*/
}
```

运行程序，输出系统日期和时间的实例效果如图 8.25 所示。

说明

time.h 是 C 语言中的一个日期和时间头文件，其提供了获取日期时间相关的函数，该文件中提供了一个 localtime 函数，用来获取一个 tm 结构表达的机器时间信息，其中包含时、分、秒、月份、年份、星期等信息；另外，还有一个比较常用的time函数，用来获取系统当前的日历时间，返回的类型为time_t类型，该类型的值为unsigned long，表示从 1970-01-01 00:00:00 到现在的秒数。

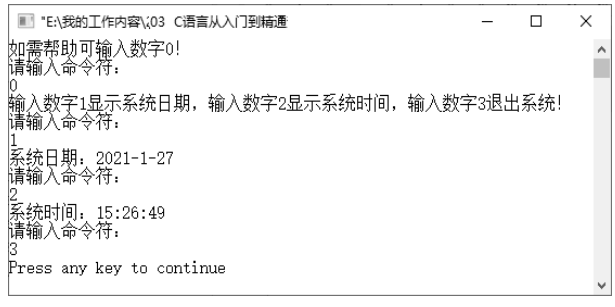


图 8.25　输出系统日期和时间

8.6.3 字符串的加密和解密

设计应用程序时，为了防止一些敏感信息泄漏，通常需要对这些信息进行加密。以用户登录密码为例，如果密码以明文形式存储在数据表中，很容易被人发现；如果密码以密文形式存储，即使别人从数据表中发现了密码，也是加密之后的密码，根本不能够使用，因此能提高系统的安全性。

【例 8.16】字符串的加密和解密（**实例位置：资源包\TM\sl\08\16**）

设计一个加密和解密算法，对一个指定的字符串加密后，再利用解密函数对密文解密，显示明文信息。加密的方式是将字符串中每个字符加上它在字符串中的位置和一个偏移值 5。以字符串“mrsoft”为例，第一个字符 m 在字符串中的位置为 0，那么它对应的密文是'm'+0+5，即 r。

在 main 函数中使用 while 语句设计一个无限循环，并声明两个字符数组，用来保存明文和密文字符串。在首次循环中，要求用户输入字符串，将明文加密成密文，之后的操作则是根据用户输入的命令字符进行判断：输入 1 加密新的明文，输入 2 对之前加密的密文进行解密，输入 3 退出系统。

```
#include<stdio.h>                                   /*包含头文件 stdio.h*/
#include<string.h>                                  /*包含头文件 string.h*/

int main()
{
    int result = 1;
    int i;
    int count = 0;
    char Text[128] = {'\0'};                        /*定义一个明文字符数组*/
    char cryptograph[128] = {'\0'};                 /*定义一个密文字符数组*/
    while (1)                                       /*通过 while 循环，使系统始终等待用户输入*/
    {
        if(result == 1)                             /*如果用户输入 1，加密明文*/
        {
            printf("请输入要加密的明文：\n");       /*输出提示信息*/
            scanf("%s", &Text);                     /*获取输入的明文*/
            count = strlen(Text);
            for(i=0; i<count; i++)                  /*遍历明文字符串*/
            {
                cryptograph[i] = Text[i] + i + 5;   /*加密后得到密文字符串*/
            }
            cryptograph[i] = '\0';                  /*设置字符串结束标记*/
            printf("加密后的密文是：%s\n",cryptograph);  /*输出密文信息*/
        }
        else if(result == 2)                        /*如果用户输入 2，解密字符串*/
        {
            count = strlen(Text);
            for(i=0; i<count; i++)                  /*遍历密文字符串*/
            {
                Text[i] = cryptograph[i] - i - 5;   /*解密后得到明文字符串*/
            }
            Text[i] = '\0';                         /*设置字符串结束标记*/
            printf("解密后的明文是：%s\n",Text);    /*输出明文信息*/
        }
        else if(result == 3)                        /*如果用户输入 3，退出系统*/
        {
            break;                                  /*跳出循环*/
        }
        else
```

```
            {
                printf("请输入命令符：\n");                    /*输出提示信息*/
            }
            printf("输入 1 加密新的明文，输入 2 对刚加密的密文进行解密，输入 3 退出系统：\n");
            printf("请输入命令符：\n");                        /*输出提示信息，提示用户输入指令*/
            scanf("%d", &result);                              /*获取输入的命令字符*/
        }
        return 0;                                              /*程序结束*/
}
```

运行程序，字符串的加密和解密效果如图 8.26 所示。

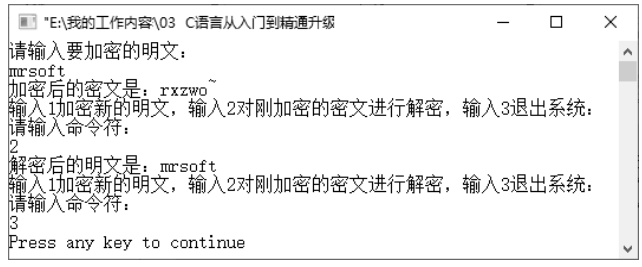


图 8.26　字符串的加密和解密

编程训练（答案位置：资源包\TM\sl\08\编程训练\）

训练 9：升序排列字符串　将已按升序排好的字符串 a 和字符串 b 按升序归并到字符串 c 中，并输出。运行结果如下：

```
请输入排序好的 a:
abcegiklnt
请输入排序好的 b:
dfmopsyz
输出排序好的 c:
abcdefgiklmnopstyz
```

训练 10：竞选班长　班级竞选班长，共有 3 个候选人。输入参加选举的人数及每个人推举的候选人编号，输出 3 个候选人最终的票数及无效选票数。输出结果如下：

```
请输入参加投票人数:
15
输入每个人推举的候选人（输入 1、2、3 即可）
1 2 3 2 2 3 3 5 6 1 1 2 2 3 1
结果如下:
候选人 1:4
候选人 2:5
候选人 3:4
无效票数:2
```

8.7　实践与练习

（答案位置：资源包\TM\sl\08\实践与练习\）

综合练习 1：十二星座速配　巨蟹座和十二星座的爱情匹配分值（星座名/速配值）为：白羊座 50；金牛座 90；双子座 70；巨蟹座 80；狮子座 75；处女座 89；天枰座 55；天蝎座 100；射手座 40；摩羯座 60；水瓶座 45；双鱼座 99。利用插入排序法，将 12 个速配值从小到大排序，运行结果如下：

```
巨蟹座与哪个星座匹配，匹配分数由低到高如下:
40      45      50      55      60
70      75      80      89      90
```

综合练习 2：统计数字出现的次数　用户输入 0～9 内任意 10 个数字，统计各数字出现的次数。运行结果如下：

```
请输入 10 个 0~9 的数组元素：
1 5 6 8 2 1 4 5 8 5
--------------------------------
|   0 出现的次数 0      |
|   1 出现的次数 2      |
|   2 出现的次数 1      |
|   3 出现的次数 0      |
|   4 出现的次数 1      |
|   5 出现的次数 3      |
|   6 出现的次数 1      |
|   7 出现的次数 0      |
|   8 出现的次数 2      |
|   9 出现的次数 0      |
--------------------------------
```

综合练习 3：成绩单公布　某班期中考试后，根据“总成绩=智育成绩*60%+德育成绩*30%+体育成绩*10%”的计算方式，将班级前 12 名同学的总成绩进行排名。输入每个人的总成绩，输出从高到低排序后的成绩。运行结果如下：

```
输入成绩：
a[0]=95
a[1]=85
a[2]=90
a[3]=77
a[4]=88
a[5]=100
a[6]=96
a[7]=93
a[8]=80
a[9]=79
a[10]=89
a[11]=92
成绩排名如下：
100     96      95      93      92
90      89      88      85      80      79      77
```

综合练习 4：平安夜卖苹果　平安夜，父亲推出一车苹果（共 2520 个），分给 6 个儿子卖。父亲先按事先写在纸上的数字把苹果分完，每个人拿到的苹果数量都不同。然后他说：“老大，把你的苹果分 1/8 给老二。老二连同原来的苹果，分 1/7 给老三。老三连同原来的苹果，分 1/6 给老四。依此类推，最后老六拿到后，连同原来的苹果分 1/3 给老大。这样，你们每个人手中的苹果就一样多了。”兄弟 6 人原来各有多少苹果呢？编写程序计算一下。输出结果如下：

```
x[1]=240
x[2]=460
x[3]=434
x[4]=441
x[5]=455
x[6]=490
```

综合练习 5：谁被@了　QQ 群或微信群里，要是找某个人有急事，就会@他。编写程序，输出被@的人员列表。输出结果如下：

```
被@的列表：
明日科技  你被@了
扎克伯格  你被@了
比尔盖茨  你被@了
```