第 3 章　流程控制语句

程序的执行并不总是按照代码先后顺序逐条进行的，程序在运行过程中，经常需要从一条语句转移到另外一条语句，这是通过流程控制语句来实现的。与所有高级语言一样，C#中的流程控制语句主要包括选择和循环控制语句。

3.1　条件控制结构

在程序执行时，经常会遇到这样的情况：如果满足某种条件就执行一些代码，如果不满足条件就执行另外一些代码。例如，一个学生成绩管理系统的某一部分代码，如果学生成绩小于 60 分，就以红色显示学生成绩，否则就以默认颜色黑色显示学生成绩。这种功能可以用条件判断语句来实现。

3.1.1　条件判断 if 语句

if 语句是最简单的条件判断语句，用来根据某个条件对程序的执行进行两路分支。if 语句的语法如下：

```
if(条件)
{
语句块 1
}
else
{
语句块 2
}
```

上述语句中的 else 部分可以有或者没有。语句的执行流程如图 3.1 所示。

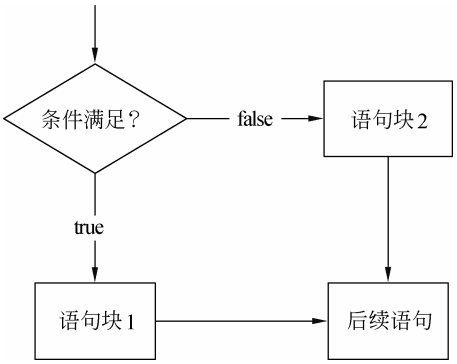

图 3.1　if…else 语句执行流程

【例 3-1】 学生成绩判断。

本例是一个判断学生成绩是否及格的控制台程序。程序让用户输入一个学生成绩，如果输入的成绩不在 0～100 之间，则给出错误提示。如果成绩在 0～59 之间，程序输出不及格。如果成绩在 60～100 之间，程序输出及格。程序代码如下。

```
static void Main(string[] args)
{
     Console.WriteLine("请输入你的成绩");
     string input=Console.ReadLine();
     int score;
     score=int.Parse(input);
     if ((score>100)||(score<0))
        Console.WriteLine("你输入的成绩不正确。必须是 0 到 100 之间的数值。");
     else if (score>60)
        Console.WriteLine("恭喜! 你及格了! ");
     else
        Console.WriteLine("你没有及格。");
     Console.ReadLine();
}
```

提示：代码中的 int.Parse(input)方法，其作用是将字符串类型转换为整型。

3.1.2 条件选择 switch…case 语句

当分支条件很多时，使用 if…else 语句可能比较麻烦，这种情况下可以使用 switch…case 语句。switch…case 语句是一个多条件分支语句。语法如下：

```
switch(控制表达式)
{
   case 常量表达式 1:
       语句块 1;
       break;
   case 常量表达式 2:
       语句块 2;
       break;
   ⋮
  case 常量表达式 n:
      语句块 n;
      break;
   default:
      默认语句块;
      break;
}
```

break 关键字表示结束 switch 语句，跳转到 switch 语句块后面的代码继续执行。switch…

case 语句的执行流程如图 3.2 所示。

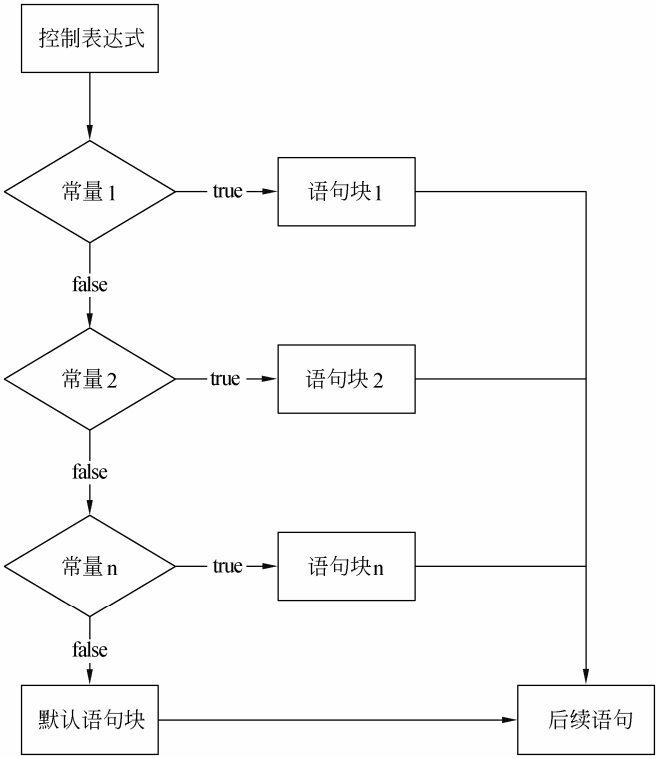

图 3.2　switch…case 语句执行流程

switch…case 语句首先计算控制表达式的值，然后用这个值和 case 标签后面的常量表达式做对比，如果找到某个 case 常量表达式与控制表达式值相同，就执行这个 case 后面的语句块。如果所有 case 标签后的常量都不等于控制表达式，那么就执行 default 标签后的语句。

switch…case 语句和 if…else 语句不同，在 if…else 语句中，总会有而且只会有一个备选语句块被执行。而在 switch…case 语句中，当控制表达式的值等于 case 标签某个常量表达式时，会执行这个 case 标签后面的语句块。如果不加 break 语句，那么程序执行完当前 case 标签的语句块后，还会继续往下执行，直到遇到 break 语句或者整个 switch 语句结束。所以，在使用 switch 语句时，一定要记得在每一个 case 标签后面写上 break 语句。

【例 3-2】 输出阿拉伯数字对应的大写数字。

这个小程序根据用户输入的每一个阿拉伯数字，输出对应的大写数字。程序代码如下。

```
static void Main(string[] args)
{
    Console.WriteLine("请输入一个数字");
    string input=Console.ReadLine();
    string digit;
    string input0=input.Substring(0, 1);//取得用户输入的第一个字符
    switch (input0)
```

```
    {
        case "0":
            digit = "零";
            break;
        case "1":
            digit = "壹";
            break;
        case "2":
            digit = "贰";
            break;
        case "3":
            digit = "叁";
            break;
        case "4":
            digit = "肆";
            break;
        case "5":
            digit = "伍";
            break;
        case "6":
            digit = "陆";
            break;
        case "7":
            digit = "柒";
            break;
        case "8":
            digit = "捌";
            break;
        case "9":
            digit = "玖";
            break;
        default:
            digit = "未知";
            break;
    }
    Console.WriteLine("与数字{0}对应的大写数字是{1}",input0,digit);
    Console.ReadLine();
}
```

3.2 循环控制结构

在编写程序的过程中，经常遇到某些需要被重复执行的功能。例如，一个学生信息管理系统，在期末考试时，需要把参加考试的学生准考证打印出来。这个打印准考证的功能，需要被执行多遍，而与此相对应的代码，也需要被执行多遍。在程序设计语言中，使用循

环结构来实现代码的重复执行。

3.2.1 while 循环

while 语句的功能是当某个条件为真时，重复执行一段代码。while 语句的语法如下：

```
while (条件)
{
    语句块
}
```

while 关键字后面括号中的条件是一个布尔表达式，当这个表达式的值为 true 时，将执行大括号内的语句块，然后再次判断条件表达式的值。如果为 true，再执行语句块，如此重复，直到条件表达为 false，程序将往下执行 while 语句块后面的代码。while 语句对应的流程图如图 3.3 所示。

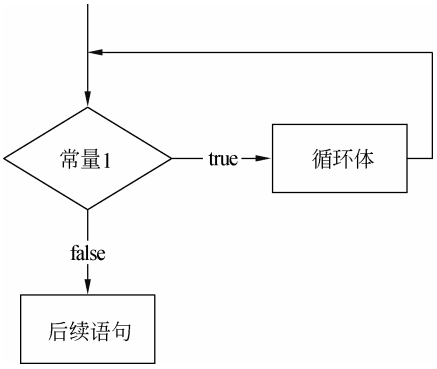

图 3.3 while 语句执行流程

一般来说，在 while 循环体内应该有影响判断条件的值的语句。如果不是这样，那么判断条件的值将永远为真，从而程序就形成死循环。例如，如果判断条件是 i>0，那么在循环体中，应该有语句对 i 的值进行修改，这样才有可能使 i 的值发生改变，才有可能退出循环。

【例 3-3】 求一个自然数的所有因子。

如果两个自然数 a 和 b 的乘积等于另外一个自然数 c，那么就说 a 和 b 是 c 的因子。显然，因子是成对出现的，例如 2×3=6，1×6=6。在两个成对的因子里面，一般都是一个因子大，一个因子小。只有一种特殊情况除外，就是当一个自然数 n 为另一个自然数 m 的平方时，这时一对因子是相等的。

通过以上分析可知，要求一个自然数的所有因子，只需要从 1 开始到 n 的平方根，逐个检测 n 能否被其整除，如果能整除，那么这个数及对应的商都是 n 的因子。程序流程图如图 3.4 所示。

下面根据刚才分析得到的算法编写一个程序来实现本例要求的功能。

（1）从 Visual Studio 2013 中创建一个控制台应用程序，将项目名称命名为 Factor。

（2）在项目的 Program.cs 文件的 Main 方法中，编写如下代码。

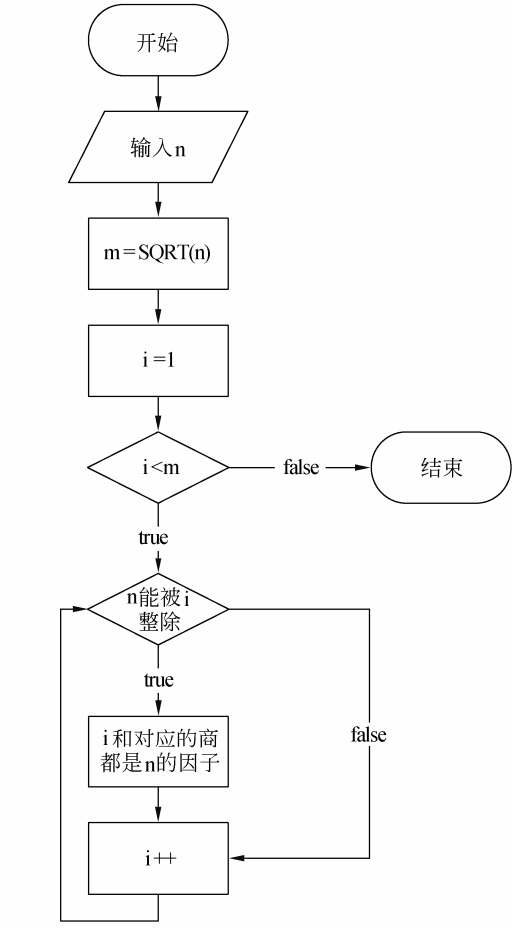

图 3.4　求自然数因子的流程

```
static void Main(string[] args)
{
    Console.Write("请输入一个自然数：");
    string input=Console.ReadLine();
    int n=int.Parse(input);
    if (n<=0)
    {
        Console.WriteLine("不能输入负数");
        return;
    }
    int i=1;
    while(i<=Math.Sqrt(n))
    {
        if(n%i==0)
        {
            Console.WriteLine(i);
            Console.WriteLine(n/i);
```

```
            }
            i++;
        }
        Console.WriteLine("按 Enter 键结束程序");
        Console.ReadLine();
    }
```

（3）按 F5 键或者单击调试工具栏上的“启动”按钮，运行应用程序。程序的运行结果如图 3.5 所示。

3.2.2 do…while 循环

do…while 循环是一个与 3.2.1 节所介绍的 while 循环很相似的循环控制结构。do…while 的语法如下：

```
do
{
语句块
}
while(条件)
```

do…while 循环的执行流程如图 3.6 所示。

图 3.5　自然数因子程序运行结果

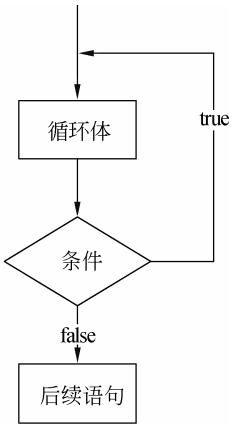

图 3.6　do…while 循环的执行流程

从图 3.6 中可以看出，do…while 循环是先执行循环体，再判断循环条件。因此，在任何情况下，do…while 循环的循环体都至少被执行一次，这也是与 while 循环的区别所在。

【例 3-4】 求学生的平均成绩。

用一个控制台应用程序来计算学生的平均成绩。程序运行后，提示用户输入学生成绩，输入一个学生成绩后，按 Enter 键，然后再输入下一个学生成绩。输入一个负数表示学生成绩输入完毕。此例的代码如下：

```
static void Main(string[] args)
{
```

```
    double mark=0;
    double sum=0;
    int count=0;
    do
    {
        Console.WriteLine("请输入学生成绩，输入负数表示结束。");
        mark=double.Parse(Console.ReadLine());
        if (mark>=0)
        {
            sum+=mark;
            count++;
        }
    }
    while (mark>=0);
    Console.WriteLine("平均成绩是{0}/{1}={2}", sum, count, Math
    .Round(sum/count,2));
    Console.ReadLine();
}
```

程序运行结果如图 3.7 所示。

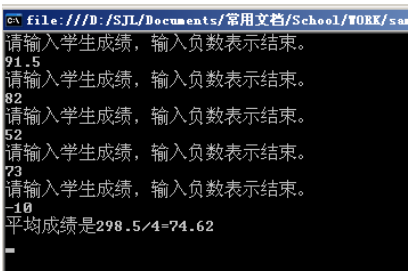

图 3.7　学生平均成绩程序运行结果

3.2.3　for 循环

for 循环是经常使用的一种循环控制结构。for 循环的语法如下：

```
for(初始化语句;循环条件;循环更改语句)
{
    循环体
}
```

for 循环与前面所介绍的 while 和 do…while 循环相比，语法上显得稍微有些复杂，因为 for 后面的括号中有用分号隔开的三条语句。其实 for 循环与 while 循环的功能是完全等价的，用 for 实现的循环完全可以用 while 来改写，同样，用 while 实现的循环也可以用 for 来改写。通过对 for 循环的执行流程进行分析，能够更好地体会这两种循环在本质上的共同之处。for 语句的执行流程如图 3.8 所示。

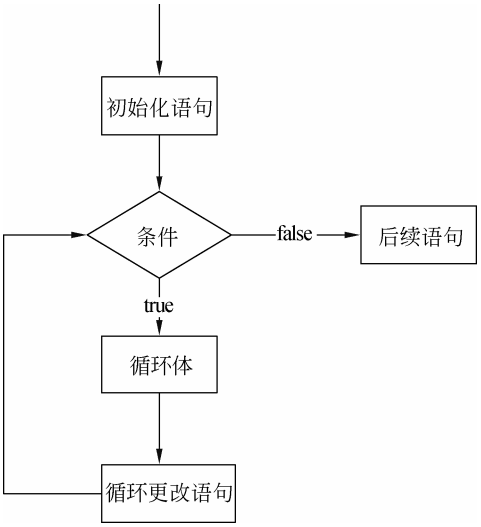

图 3.8　for 循环的执行流程

对比 for 和 while 循环的执行流程图可以看出，for 循环等价于在 while 循环的判断条件前面加一条初始化语句，并且把 for 循环中的循环体与 for 循环中的循环更改语句合并成为 while 循环中的循环体。用代码来表示，本节开始时列出的 for 循环等价于如下的 while 代码。

```
初始化语句;
while(循环条件)
{
  循环体;
  循环更改语句;
}
```

for 循环与 while 循环在功能上是等价的，但是语法不一样，而且应用的场合也有所不同。for 循环一般用于循环次数固定的循环，最经常使用的 for 循环就是类似于 for(i=0;i++;i<10){/*code*/}这种循环。而 while 循环一般在只知道循环条件，而循环的次数不固定的情况下使用。

> **注意**：for 关键字后面括号中的 3 个语句都是可选的，但是两个分号不可以省略。

【例 3-5】 求阶乘。

一个自然数 N 的阶乘定义为从 1～N 的所有自然数的乘积。在这个例子中，编写一段小程序来求一个给定自然数 N 的阶乘。这是一个典型的应用 for 循环的例子，因为循环次数是固定的，就是从 1～N。程序代码如下：

```
static void Main(string[] args)
{
      Console.Write("请输入一个自然数 N：");
      string input=Console.ReadLine();
```

```
        int n = int.Parse(input);
        int factorial=1;
        for (int i=1;i<=n;i++)
        {
            factorial*=i;
        }
        Console.WriteLine("{0}的阶乘是{1}",n,factorial);
        Console.WriteLine("按 Enter 键结束程序");
        Console.ReadLine();
}
```

程序运行结果如图 3.9 所示。

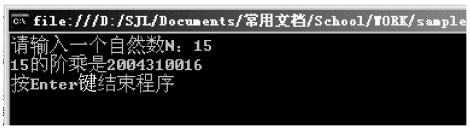

图 3.9　阶乘程序运行结果

注意：for 循环、while 循环以及 do…while 循环是等价的，任何一种循环都可以用任意另外一种循环来实现同样的功能。感兴趣的读者可以自行写出用一种循环来表示另一种循环的代码。

3.2.4　foreach 循环

在 C#中，有一些对象是由 0 到多个类型相同的对象组成的，可将其称为集合类。最常见的集合类就是数组，除此之外还有 ArrayList、Queue、DataTableCollection 类等也都是集合类。有时需要对集合类中的每个元素逐一进行某种操作，在这种情况下，通常使用 foreach 循环来完成这项功能。foreach 循环的语法结构如下：

```
foreach([类型] 变量名 in 集合)
{
   循环体
}
```

foreach 循环的作用是对于集合中的每个元素，逐个执行循环体的内容。图 3.10 所示为 foreach 循环的执行流程图。

【例 3-6】 找出一个英文句子中的单词。

这个小程序的功能是提示用户输入一个英文句子，然后找出其中包含的单词并且输出。在英语中，单词之间是由空格或者标点符号隔开的。假设用户输入的句子语法及拼写是正确的，那么只要以空格或者标点符号为间隔，把整个句子截成若干部分，每一部分就是一个英语单词。

.NET Framework 提供了强大的字符串处理功能，其中 String 类的 Split 方法可以根据指定的分隔符，把一个字符串拆分成一个字符串数组。这个功能应用在本例中非常合适。

新建一个控制台应用程序，将项目命名为 FindWords，并在 Main 方法中输入以下代码。

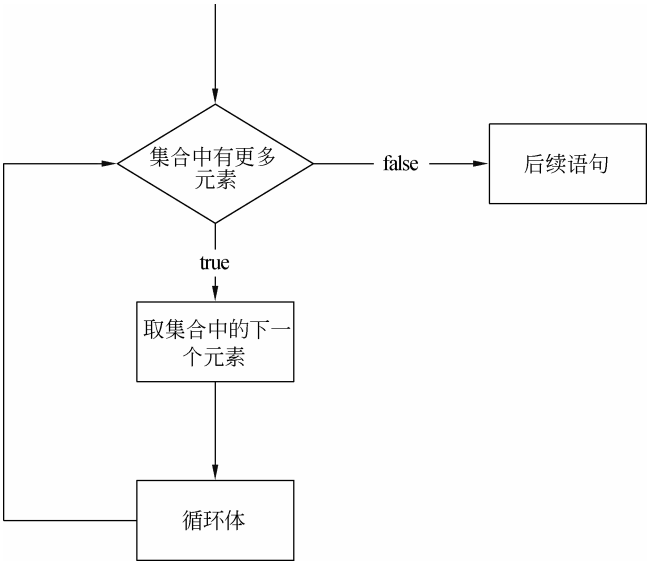

图 3.10　foreach 循环的执行流程

```
//得到一个英文句子中的所有单词
static void Main(string[] args)
{
    Console.WriteLine("请输入一个英文句子");
    string sentence=Console.ReadLine();
    //以常用标点为分隔符，把整个句子分割成字符串数组
    string[] words=sentence.Split(' ', ',', ';','.');
    //输出数组中的单词
    Console.WriteLine("此句子中包含以下单词：");
    foreach(string word in words)
    {
        Console.WriteLine(word);
    }
    Console.WriteLine("按 Enter 键结束程序");
    Console.ReadLine();
}
```

程序运行结果如图 3.11 所示。

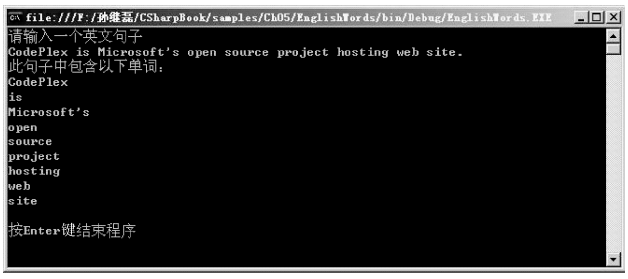

图 3.11　FindWords 程序运行结果

提示： String.Split 是一系列重载的方法，有多种不同的语法。在本例中使用到的语法是 String.Split(char 分隔字符 1，char 分隔字符 2，char 分隔字符 3，……，char 分隔字符 n)。这个方法的功能是以参数中的字符为分隔符，把字符串拆分成为一个字符串数组。String.Split 的完整语法请参见微软公司的 MSDN。

3.2.5 break 和 continue 关键字

在循环体执行过程中，在某种情况下，可能希望中断循环的执行。C#中提供了 break 关键字和 continue 关键字来实现这个功能。两者不同的是，break 关键字中断整个循环，让程序继续循环体后面的代码，而 continue 关键字只是中断当前的一次循环，让程序返回循环入口，继续执行下一轮循环。图 3.12 所示是 break 和 continue 的执行流程示意图。

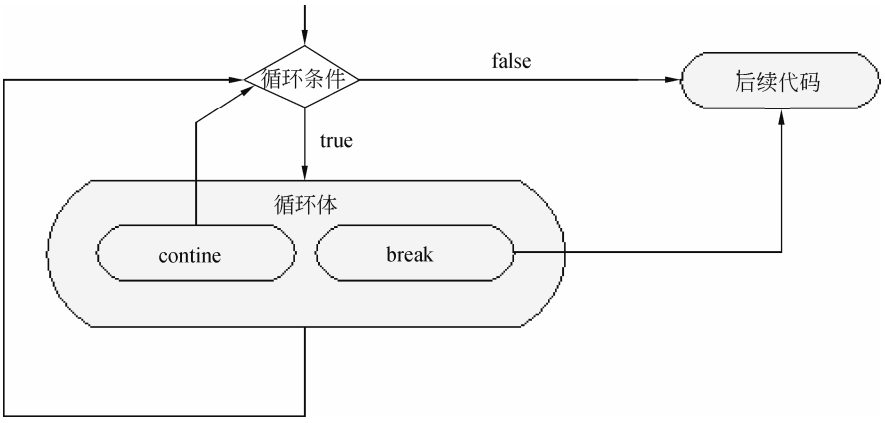

图 3.12　break 和 continue 的执行流程

【例 3-7】 求学生的平均成绩(改进版)。

本例是对例 3-4 的改进版本。在例 3-4 中，没有对用户输入的错误数据进行处理。如果输入非法数据，例如输入的不是数值，或者输入的数值不合理（如 2000），那么将导致程序运行失败。本例将修正这几个错误，对用户输入的数据进行检验，丢弃不合法的数据，从而使程序具有一定容错能力。改进后的求学生平均成绩的流程图如图 3.13 所示。

改进后的学生平均成绩程序代码如下：

```
static void Main(string[] args)
{
    double mark=0;
    double sum=0;
    int count=0;
    bool valid;
    string input;
    while(true)
    {
        Console.WriteLine("请输入学生成绩(0～100)，输入负数表示结束。");
        input=Console.ReadLine();                    //接收用户输入
        valid=double.TryParse(input,out mark);   //检查用户输入是否为数值
```

```
        if (!valid)                          //不是数值，进入下一轮循环
            continue;
        if (mark<0)                          //输入负数，结束整个循环
            break;
        else if (mark>100)                   //忽略大于100的值，进入下一轮循环
       continue;
        else                                 //输入合法，进行分数累加
        {
            sum+=mark;
            count++;
        }
    }
    Console.WriteLine("平均成绩是{0}/{1}={2}", sum, count,
    Math.Round(sum/count, 2));
    Console.ReadLine();
   }
}
```

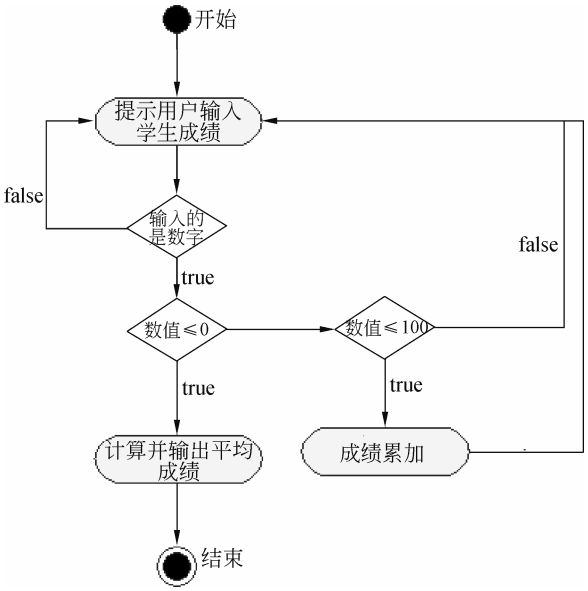

图 3.13　求学生平均成绩的流程图

程序运行结果如图 3.14 所示。

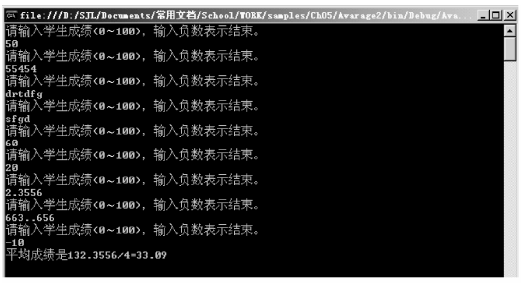

图 3.14　改进版求学生平均成绩程序的运行结果

提示：本例中用到了 Double.TryParse 方法，其语法是 Double.TryParse(string text,out double value)。这个方法的功能是，尝试把输入的字符串转换为 double 类型，如果转换成功，那么这个方法返回 true，并且以 out 方法传递的 value 参数中包含转换后的数值；如果转换失败，这个方法返回 false。与此类似的方法还有 Int32.TryParse、DateTime.TryParse 等，具体的使用方法可以参考 MSDN。

3.2.6 多重循环

多重循环指的是循环的嵌套，即在一个循环体内又嵌套了另外一个循环。多重循环在执行时，每执行一次外层循环，都要执行一遍整个内层循环。下面通过一个实际例子来说明多重循环的应用。

【例 3-8】 冒泡排序。

在编写程序的过程中，排序是经常会遇到的一个问题。排序指的是对一组可排序的数据，按照大小顺序排列起来，使其变为一个有序的序列。

排序有各种算法，冒泡排序是一种容易理解、实现简单的算法。

假设要排序的序列是一个数值序列，序列长度为 N。冒泡排序的原理是，从第一个数开始，比较它和下一个数，如果是逆序的（即第二个数大于第一个数），则把这两个数做交换。然后再比较第二个和第三个，依次类推，直到比较第 N–1 个和最后一个，如果逆序则交换。这个过程称为一趟排序。

经过上述的一趟排序以后，最大值已经排在了序列的最后，第二趟排序只需在第一个元素到第 N–1 个元素之间进行。然后再进行第三趟、第四趟，直到第 N–1 趟排序。在整个排序过程中，每一趟排序都有一个最大值“沉入”底部（序列末尾），而数据较小的值则像气泡一样逐渐“上浮”到顶部（序列开头），所以这种算法称为冒泡算法。

如果冒泡排序的某一趟排序中没有发生任何元素交换，则说明任意两个相邻元素是有序的，从而整个序列已经有序，此时可以提前退出排序程序。冒泡排序的流程图如图 3.15 所示。

接下来用 C#代码实现上述冒泡排序算法。

（1）在 Visual Studio 2013 中，创建一个新的控制台应用程序，将项目重命名为 BubbleSort。

（2）在 Visual Studio 2013 自动生成的 Program.cs 文件中，添加一个 BubbleSort 方法，方法的代码如下：

```
//冒泡排序算法，nums 为待排序的整数数组
private static void BubbleSort(int[] nums)
{
       int i, j, temp;
       bool exchange;  //交换标志
       for (i=0;i<nums.Length;i++)  //最多做 R.Length-1 趟排序
       {
           exchange=false;           //本趟排序开始前，交换标志应为假
           for (j=0;j<nums.Length-i-1;j++)
           {
```

```
            if (nums[j]>nums[j+1])    //交换这两个数字
                {
                    temp=nums[j + 1];
                    nums[j+1]=nums[j];
                    nums[j]=temp;
                    exchange=true;     //发生了交换，故将交换标志置为真
                }
            }
            if (!exchange)             //本趟排序未发生交换，提前终止算法
            {
                break;
            }
        }
    }
```

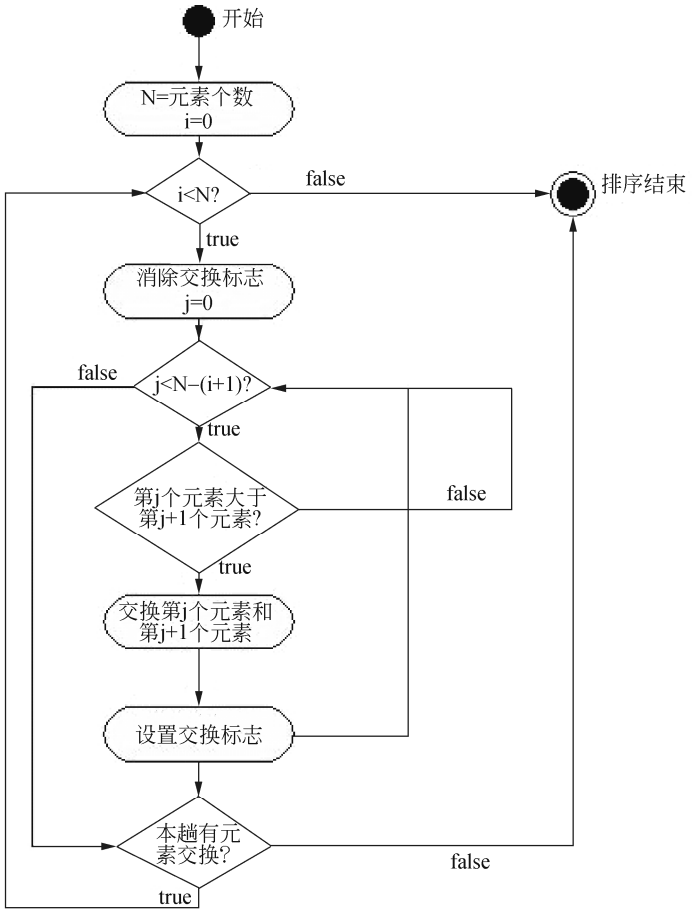

图 3.15　冒泡排序的流程图

（3）在 Program 类中添加一个 displayNums 方法，用来显示一个整型数组中的所有数据。displayNums 方法的代码如下：

```
//显示数组中的数值
private static void displayNums(int[] nums)
{
    for (int i=0;i<nums.Length;i++)
    {
        Console.Write(nums[i]);
        Console.Write("\t");
    }
}
```

（4）在 Main 方法中，编写代码，生成测试数据，调用 BubbleSort 方法和 displayNums 方法，实现冒泡排序过程并将排序结果输出。代码如下：

```
static void Main(string[] args)
{
    int[] nums=new int[]{34,554,87,90,32,452,2323,25,6,2,365,25,81,
    42,365,21,42,52};
    Console.WriteLine("排序前的数字序列");
    displayNums(nums);
    BubbleSort(nums);
    Console.WriteLine();
    Console.WriteLine("排序后的数字序列");
    displayNums(nums);
}
```

（5）按 F5 键运行程序，程序的运行结果如图 3.16 所示。

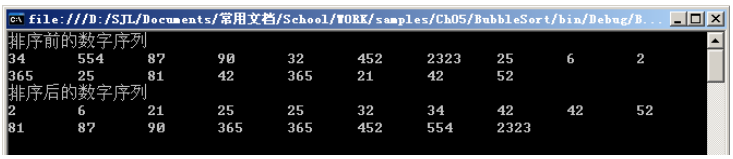

图 3.16　冒泡排序程序运行结果

3.3　数组

数组是包含多个具有相同数据类型的元素的有序集合。元素在数组中的顺序称为下标。

在编程时，经常会遇到一批类型相同、又有一定先后顺序的数据，如一个班里的学生，一批用于排序的整数，某种商品一个月以来每天的销售数量等。用数组类型可以很好地描述这些数据。

3.3.1　数组类型

C#中的数组是一个 System.Array 类型，因此数组可以使用 System.Array 类的属性和方

法。System.Array 类的主要方法和属性如表 3.1 所示。

表 3.1　System.Array 类的主要方法和属性

属性或方法名	属性/方法	静态/实例	描　　述
Length	属性	实例	得到数组的长度（数组中的元素数）
Rank	属性	实例	得到数组的维数（秩）
Clear	方法	静态	将数组中的元素设置为空值或 0
Copy	方法	静态	将源数组的指定元素复制到目的数组指定位置
CopyTo	方法	实例	将数组中的全部元素复制到目的数组指定位置
Find	方法	静态	返回第一个满足指定条件的数组元素
FindAll	方法	静态	返回数组中满足条件的所有元素
Sort	方法	静态	对数组中的元素进行排序

3.3.2　声明和分配数组

在 C#中用如下语法声明一个数组。

```
元素类型[]  数组名;    //声明一维数组
元素类型[,] 数组名;    //声明二维数组
```

例如：

```
int[] nums;          //声明一个整型数组
string[] words;      //声明一个字符串型数组
int[,] r2array;      //声明一个二维数组
```

在声明数组的同时，还可以对数组元素进行初始化，例如：

```
int[] nums=new int[]{0,1,2,3,4,5,6,7,8,9};
string[] words=new string[]{"one","two","three"};
```

可以把上面两条语句简化为

```
int[] nums={0,1,2,3,4,5,6,7,8,9};
string[] words={"one","two","three"};
```

3.3.3　数组元素访问

数组元素通过下标来进行访问，语法为

```
数组名[下标]
```

每个数组都有一定的长度，数组元素不能越界访问，即不能超过数组的最大下标。数组的下标是从 0 开始的，因此数组的合法下标是从 0～N–1（N 为数组长度）。

【例 3-9】 数组元素的访问。

创建一个控制台应用程序，在 Main 方法中输入以下代码：

```
static void Main(string[] args)
```

```
{
    Random r=new Random();
    double[] nums=new double[10];
    for (int i=0;i<10; i++)
    {
        nums[i]=r.Next();  //为第 i 个元素赋一个随机数
    }
    //逐个输出数组元素
    for (int i=0; i<10; i++)
    {
        Console.WriteLine("数组的第{0}个元素的值是{1}",i,nums[i]);
    }
    Console.ReadLine();
}
```

说明：程序中使用的 Random 类是一个用于产生随机数的类。Random.Next()方法返回一个大于或等于零，且小于 Int32.MaxValue 的整数。

按 F5 键运行程序，输出结果如下：

```
数组的第 0 个元素的值是 1927159025
数组的第 1 个元素的值是 1628618202
数组的第 2 个元素的值是 2106527823
数组的第 3 个元素的值是 1830718131
数组的第 4 个元素的值是 615201411
数组的第 5 个元素的值是 1720024621
数组的第 6 个元素的值是 909307591
数组的第 7 个元素的值是 325599139
数组的第 8 个元素的值是 950332925
数组的第 9 个元素的值是 1161888554
```

3.4 本章小结

本章介绍了 C#中的流程控制语句，主要包括分支（条件）控制和循环控制。简单分支可以用 if…else…语句实现，当分支很多时，可以用 switch…case 语句来实现。C#中经常使用的循环控制结构包括 while、do…while、for、foreach。其中前 3 个循环控制是等价的，彼此可以相互替换。但在特定的环境中，可能使用某一种循环结构比其他的更直观，更容易理解。数组是包含多个具有相同数据类型的元素的有序集合。在 C#中，数组属于 System.Array 类型。数组元素通过其下标来访问。