第 3 章　数组、字符串和集合

情景导入

在控制台学生成绩管理系统中，学生信息需要保存起来以便进行后续操作，如计算平均成绩、修改成绩、成绩统计等。显而易见的做法就是把这些数据保存到变量中，然后再读取变量的值。使用这种方式处理需要声明很多变量。在一个程序中书写很多变量声明很烦琐，数组为我们提供了声明一组相关变量的优雅方法。

学习目标

在学习完本章内容后，读者将能够：

- 定义和使用一维数组来组织数据。
- 定义和使用多维数组来组织数据。
- 选择合适的字符方法完成字符串处理。
- 列举常用集合。
- 选择合适集合组织数据。

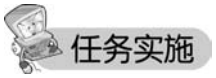
视频讲解

3.1　数　　组

任务描述

在控制台学生成绩管理系统中，用户不仅需要输入学生信息，而且也需要输出学生成绩或者统计学生成绩。本任务进一步完善控制台学生成绩管理系统的开发，完成学生信息的输入输出，如图 3-1 所示。

图 3-1　控制台学生成绩管理系统信息输出



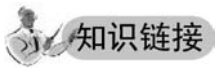
(1) 新建项目。启动 Visual Studio 2017,新建控制台程序 GradeManagement。

(2) 修改代码。项目初始化后,修改 Program.cs 文件为如下代码:

```
using System;
using static System.Console;
namespace GradeManagement
{
    class Program
    {
        static void Main(string[] args)
        {
            const int NUM = 3;                //学生人数
            //声明二维数组,存放学生信息
            string[,] student = new string[NUM, 7];
            //方法调用
            InputStudents(student, NUM);
            OutputStudents(student, NUM);
            Console.ReadKey();
        }
        /// <summary>
        ///输入学生信息
        /// </summary>
        static void InputStudents(string[,] student, int num)
        {
            Console.Clear();

            for (int i = 0; i < num; i++)
            {
                WriteLine("请输入第{0}个学生的学号: ", i + 1);
                Write("学号: ");
                student[i, 0] = Console.ReadLine();
                Write("姓名: ");
                student[i, 1] = Console.ReadLine();
                Write("语文: ");
                student[i, 2] = Console.ReadLine();
                Write("数学: ");
                student[i, 3] = Console.ReadLine();
                Write("英语: ");
                student[i, 4] = Console.ReadLine();
                //计算总分
                int temp = Convert.ToInt32(student[i, 2]) + Convert.ToInt32(student[i, 3]) +
                           Convert.ToInt32(student[i, 4]);
                student[i, 5] = Convert.ToString(temp);
                student[i, 6] = string.Format("{0:F2}", temp / 3.0);
            }
        }
        /// <summary>
        ///输出学生信息
```

```
        /// </summary>
        static void OutputStudents(string[,] student, int num)
        {
            //输出学生成绩
            Clear();
            WriteLine("\t\t\t 学生成绩单");
            WriteLine("\t\t\t      制表时间: " + DateTime.Now.ToShortDateString());
            WriteLine("|-----------------------------------------------|");
            WriteLine("| 学      号 | 姓名 |语文|数学|英语|总 分|平均分|");
            WriteLine("|-----------------------------------------------|");
            for (int i = 0; i < num; i++)
            {
                WriteLine("|{0,8}|{1,3}|{2,4}|{3,4}|{4,4}|{5,5}|{6,6:F2}|",
                          student[i, 0], student[i, 1], student[i, 2], student[i, 3],
                          student[i, 4], student[i, 5], student[i, 6]);
                WriteLine("|-----------------------------------------------|");
            }
        }
    }
}
```

(3) 编译和运行程序。按 Ctrl+F5 组合键运行该程序,运行结果如图 3-1 所示。



3.1.1 数组基础

数组是一个存储相同类型元素的固定大小的顺序集合。数组是用来存储数据的集合,通常认为数组是一个同一类型变量的集合。例如,一组由 7 个 double 类型变量组成的数组可以按照下面的方法创建:

```
double[ ] temperature = new double[7];
```

这类似于声明下述 7 个有点怪异、类型为 double 的变量: temperature[0]、temperature[1]、temperature[2]、temperature[3]、temperature[4]、temperature[5]、temperature[6]。像这样在方括号[]中放置一个整数表达式的变量称为下标变量、索引变量或者数组元素。方括号中的整数表达式称为索引或者下标。注意,下标的编号是从 0 开始计数的,而不是从 1 开始计数的。

这 7 个变量中的每一个变量都可以像任何其他 double 类型的变量那样使用。例如,下面所有语句在 C# 中都是允许的:

```
temperature[3] = 32;
temperature[6] = temperature[3] + 5;
Console.WriteLine(temperature[6]);
```

当把这些下标变量看作一个整体,当作一个数据项时,就称它们为数组。在数组中,下标是数组元素的一部分,它并不一定是整数常量,也可以是任何整数表达式。由于下标可以是表达式,因此可以编写一个循环把各个值读入到数组 temperature 中:

```
Console.WriteLine("请输入一周的温度: ");
for(int index = 0 ; index < 7 ; index++)
    temperature[index] = Convert.ToDouble(Console.ReadLine());
```

用户可以在多行上(中间用回车符分隔)输入 7 个值。在读入数组值后,可以使用如下方式显示它们:

```
Console.WriteLine("一周的温度情况如下: ");
for (int index = 0 ; index < 7 ; index++)
    Console.Write(temperature[index] + " ");
Console.WriteLine();
```

1. 声明和创建数组

可以使用 new 运算符来创建数组。当创建元素类型为 Base_Type 的数组时,语法如下:

```
Base_Type[ ] Array_Name = new Base_Type[Length];
```

例如,下面的语句创建一个名为 pressure 的数组,它等价于 100 个 int 类型变量:

```
int[ ] pressure = new int[100];
```

作为一种替代的方法,前面的语句也可以拆分为两个步骤:

```
int[ ] pressure;
pressure = new int[100];
```

第一个步骤声明一个整数数组,第二个步骤为数组分配足够容纳 100 个整数的内存空间。数组的类型称为数组的基本类型,数组的基本类型可以是任何数据类型。在这个示例中,基本类型为 int。数组中元素的个数称为数组的长度、容量或者大小。因此,这个样本数组 pressure 的长度为 100,这意味着它具有下标变量 pressure[0]～pressure[99]。注意,由于下标从 0 开始,长度为 100 的数组(例如 pressure)没有下标变量 pressure[100]。

2. 初始化数组

数组可以在声明时初始化。要完成这个任务,可以把各个下标变量的值放在大括号{}中,并把它们放在赋值运算符=的后面,如下所示:

```
double[ ] reading = {3.3, 15.8, 9.7};
```

数组的大小(也就是它的长度)被设置为可以容纳给定值的最小值(例如 3)。因此,上述初始化声明等价于下述语句:

```
double[ ] reading = new double[3];
reading[0] = 3.3;
reading[1] = 15.8;
reading[0] = 9.7;
```

如果没有对数组进行初始化,它们或许会自动初始化为基本类型的默认值。例如,如果没有对整数数组进行初始化,那么数组的第一个元素将会被初始化为 0。但是,明确初始化数组将会使程序更为清晰。可以使用这两种方法中的一种初始化数组:或者使用大括号,就像前面描述的那样,直接将值读入数组元素;或者通过赋值,就像下面的 for 循环语句中

所做的那样。

```
int[] count = new int[100];
for (int i = 0; i < 100;i++)
  count[i] = 0;
```


【编程示例】 使用随机数生成一个含有 10 个元素的整数数组，接收插入数据的位置和数据后，将数据插入到指定位置，然后输出新数组。新建控制台程序 UseArray，代码初始化后，在 Program.cs 文件中添加如下代码：

```
static void Main(string[] args)
{
    int[] myIntArray = new int[11];                          //声明数组
    Random ram = new Random();                               //随机数对象
    //利用循环完成数组初始化
    for (int i = 0; i < 10; i++)
    {
        //随机生成[10,100]的整数
        myIntArray[i] = ram.Next(9, 99) + 1;
    }
    //显示数组
    DispalyArray(myIntArray, "before");
    Write("\n 输入插入位置(0-9): ");
    int pos = Convert.ToInt32(ReadLine());
    Write(" 输入插入数据: ");
    int number = Convert.ToInt32(ReadLine());
    for (int k = myIntArray.Length - 1; k > pos; k-- )       //数组元素移位
    {
        myIntArray[k] = myIntArray[k - 1];
    }
    myIntArray[pos] = number;
    //输出插入数据后的数组
    DispalyArray(myIntArray, "after");
    Console.ReadKey();
}
//显示数组
static void DispalyArray(int[] array,string when)
{
    WriteLine("Array values " + when + " inserting");
    foreach (int m in array)                                 //遍历数组
    {
        Write("{0,4}", m);                                   //输出数组元素的值
    }
}
```

按 Ctrl+F5 组合键运行该程序，结果如图 3-2 所示。

3.1.2 多维数组

具有多个下标的数组有时也很有用。例如，在学生成绩管理系统中，如果要保存 50 个学生的信息，可以使用一个下标表示学生所在行，用另外一个下标表示学生的某个数据项，

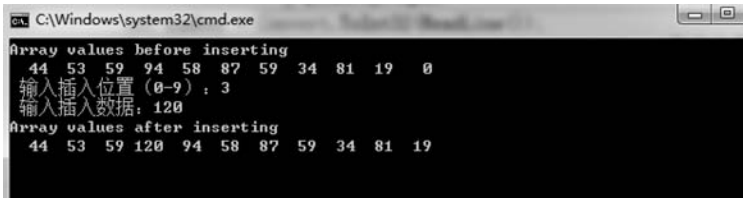


图 3-2　示例程序的运行结果

这样就可方便地跟踪所有的学生信息。包含两个下标的数组称为二维数组。二维数组元素的 C＃表示方法如下：

```
Array_Name[Row_index, Column_index]
```

例如，如果数组名为 table，并且具有两个下标，那么 table[2,2]是行下标为 2、列下标为 2 的数据项。

具有多个下标的数组通常称为多维数组，它们几乎可以采用与一维数组相同的方式处理。典型地，在 C＃程序中，声明和创建一个二维数组的语法如下：

```
Base_Type[,] Array_Name = new Base_Type[Length_1,Length_2];
```

下面的语句声明名为 student 的数组，并创建它：

```
string[,] student = new string[3,7];
```

这个声明等价于下述两条语句：

```
string[,] student;
student = new string[3,7];
```

注意，这个语法几乎与用于一维数组的语法相同。唯一的差别是，这里有两个下标。与一维数组中的下标变量相似，多维数组中的下标变量也是基本类型的变量，并且可以在具有基本类型变量允许使用的任何地方使用。

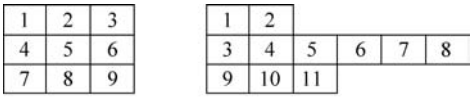
多维数组的初始化和一维数组类似，可以使用关键字 new 来动态初始化数组或者通过给定值的形式来指定数组中的全部内容。例如：

```
double[,] hillHeight = new double[3,4]{{1,2,3,4},{2,3,4,5},{3,4,5,6}};
double[,] hillHeight = {{1,2,3,4},{2,3,4,5},{3,4,5,6}};
```

3.1.3　交错数组

前面介绍的多维数组每一行的元素都相同，因此可称为矩形数组。如果多维数组中每一行的元素个数都不同，这样就构成了交错数组。交错数组被称为数组中的数组，因为它的每个元素都是另一数组。图 3-3 比较了有 3×3 个元素的二维数组和交错数组。图 3-3(b)中的交错数组有 3 行，第一行有 2 个元素，第二行有 6 个元素，第三行有 3 个元素。

在声明交错数组时，要依次放置开闭括号。在初始化交错数组时，先设置该数组包含的行数。定义各行中元素个数的第二个括号设置为空，因为这类数组的每一行都包含不同的元素数。然后，为每一行指定行中的元素个数。例如下面的代码：



(a) 二维数组　　(b) 交错数组

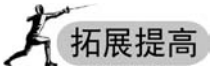
图 3-3　二维数组与交错数组示例

```
int[][] jagged = new int[3][];
jagged[0] = new int[2] {1, 2};
jagged[1] = new int[6] {3, 4, 5, 6, 7, 8};
jagged[2] = new int[3] {9, 10, 11};
```

迭代交错数组中所有元素的代码可以放在嵌套的 for 循环语句中。在外层的 for 循环语句中，迭代每一行，内层的 for 循环语句迭代一行中的每个元素，例如：

```
for ( int row = 0; row < jagged.Length; row++)
{
for ( int element = 0; element < jagged[row].Length; element++)
{
    Console.WriteLine("row: {0}, element: [1], value: {2}",row, element, jagged[row].
[element]);
}
}
```

在某些情况下，使用不规则的交错数组比较有利，但是，绝大多数应用程序不需要它们。然而，如果理解了交错数组，将可以更好地理解多维数组在 C# 中是如何工作的。



1. Array 类

Array 类是 C# 中所有数组的基类，它是在 System 命名空间中定义的。Array 类提供了各种用于数组的属性和方法。Array 类的主要方法如下。

(1) Indexof(Array array,Object)：返回第一次出现的下标。

(2) Sort(Array array)：从小到大排序（仅支持一维数组）。

(3) Reverse(Array array)：数组逆置。

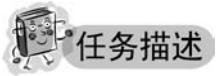
(4) Clear(Array array,int index,int length)：将某个范围内的所有元素设为初始值。

2. 编程实践

编写一个 Windows 窗体应用程序，在上面添加两个按钮，一个用来生成随机数组，一个用来排序，单击“生成随机数组”按钮，生成一个随机数组，并将数据显示在上方的输入框中，然后单击“排序数组”按钮，则对数组进行排序，并显示在下方的文本框中。

视频讲解

3.2　字　符　串



在控制台学生成绩管理系统中，用户往往希望一次性地输入学生信息，然后再对数据进行处理和输出。本任务进一步优化学生成绩管理系统的开发，实现学生信息整体输入，如图 3-4 所示。

图 3-4　学生信息整体输入

任务实施

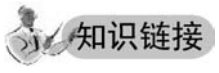
（1）新建项目。启动 Visual Studio 2017，新建控制台项目 InputStudent。

（2）修改代码。项目初始化以后，在主窗口显示的 Program.cs 文件中添加如下代码行：

```
static void Main(string[] args)
{
    const int NUM = 3;                                  //学生人数
    string[,] student = new string[NUM, 7];             //二维数组声明
    InputStudent(student, NUM);                         //调用学生信息输入方法
    OutputStudent(student, NUM);                        //调用学生信息输出方法
}
//学生信息输入方法
static void InputStudent(string[ ,] student, int num)
{
    int temp;
    string strStudent = string.Empty;                   //初始时字符为空
    string[] strInfo;
    for (int i = 0; i < num; i++)                       //输入学生信息
    {
        Console.Write("输入第{0}个学生信息(以顿号或空格分隔)：", i + 1);
        strStudent = Console.ReadLine();
        strInfo = strStudent.Split('、');               //分隔字符串
        for (int j = 0; j < strInfo.Length; j++)
        {
            student[i, j] = strInfo[j];
        }
        //计算总分
        temp = Convert.ToInt32(student[i, 2]) + Convert.ToInt32(student[i, 3]) + Convert.
        ToInt32(student[i, 4]);
        student[i, 5] = Convert.ToString(temp);
        student[i, 6] = string.Format("{0:F2}", temp/3.0);
    }
}
//学生信息输出
static void OutputStudent(string[,] student, int num)
{
```

```
        //输出学生成绩
        Console.WriteLine("                    学生成绩单");
        Console.WriteLine("|-----------------------------------------|");
        Console.WriteLine("| 学  号 | 姓名 |语文|数学|英语|总 分|平均分|");
        Console.WriteLine("|-----------------------------------------|");
        for (int i = 0; i < num; i++)
        {
            //格式化字符串
            string tempString = string.Format("|{0,8}|{1,3}|{2,4}|{3,4}|{4,4}|{5,5}|{6,6:f2}|",
                                student[i, 0], student[i, 1], student[i, 2],
                                student[i, 3], student[i, 4], student[i, 5],
                                student[i, 6]);
            Console.WriteLine(tempString);
            Console.WriteLine("|-----------------------------------------|");
        }
    }
```

(3) 编辑和运行程序。按 Ctrl+F5 组合键运行该程序,运行结果如图 3-4 所示。



3.2.1 字符串基础

字符串是 C# 最重要的数据类型之一,是.NET Framework 中 String 类的对象。String 对象是 System.Char 对象的有序集合,它的值是该有序集合的内容,而且该值是不能改变的。System.Char 类只定义了一个 Unicode 字符。Unicode 字符是目前计算机中通用的字符编码,它为不同语言中的每个字符都设定了唯一的二进制编码,用于满足跨平台、跨语言的文本转换和处理要求。

由于字符串是由零个或多个字符组成的,可以根据字符在字符串的索引值来获取字符串中的某个字符。字符在字符串中的索引从 0 开始。例如,字符串"Hello C#!"中第一个字符是 H,它在字符串中的索引值为 0。字符串中可以包含转义字符对其中的内容进行转义,也可以在前面加@符号使其中的所有内容不再进行转义。例如:

```
string path = "C:\\xcu\\xcu.java";                    //使用转义字符
string path = @"C:\xcu\xcu.java";                     //前面加上@
```

在 C# 中,string 关键字是 System.String 类的别名。因此,String 与 string 等效,用户可以根据自己的喜好选择命名约定。在程序开发中,可以通过以下方式来创建字符串:

(1) 通过给 String 变量指定一个字符串,例如:

```
string oldPath = "c:\\Program Files\\Microsoft Visual Studio 13.0";    //用转义字符初始化字符串
string newPath = @"c:\Program Files\Microsoft Visual Studio 9.0";      //用@来实现转义字符
```

(2) 使用 String 类构造函数,例如:

```
char[] letters = { 'H', 'e', 'l', 'l','o' };
string greetings = new string(letters);
Console.WriteLine("Greetings: {0}", greetings);
```

(3) 通过使用字符串串联运算符(+),例如:

```
string fname = "Rowan";
string lname = "Atkinson";
string fullname = fname + lname;
Console.WriteLine("Full Name: {0}", fullname);
```

3.2.2 字符串常用操作

1. 比较字符串

比较字符串时,产生的结果会是一个字符串大于或小于另一个字符串,或者两个字符串相等。根据执行的是序号比较还是区分区域性比较,确定结果时所依据的规则会有所不同。对特定的任务使用正确类型的比较十分重要。在 String 类中,常见的比较字符串的方法有 Compare()、CompareTo()、CompareOrdinal()以及 Equals()等。

1) Compare()方法

Compare()方法是 String 类的静态方法,用于全面比较两个字符串对象。它有多种重载方式,其中最常用的两种方法如下:

```
int Compare(string strA, string strB)
int Compare(string strA, string strB, bool ignoreCase)
```

其中,strA、strB 为待比较的两个字符串,ignoreCase 指定是否考虑大小写,当其值取 true 时忽略大小写。表 3-1 给出了 Compare()方法可能的返回值。

表 3-1 Compare ()方法可能的返回值

返 回 值	条 件
负整数	在排序顺序中,第一个字符串在第二个字符串之前,或者第一个字符串是 null
0	第一个字符串和第二个字符串相等,或者两个字符串都是 null
正整数	在排序顺序中,第一个字符串在第二个字符串之后或者第二个字符串是 null

下面的示例使用 Compare()方法来确定两个字符串的相对值。

```
string MyString = "Hello World!";
Console.WriteLine(String.Compare(MyString, "Hello World?"));  //此示例向控制台显示 -1
```

2) CompareTo()方法

CompareTo()方法将当前字符串对象与另一个字符串或字符串对象进行比较,并返回一个整数。该整数指示此字符串对象在排序顺序中的位置是位于指定字符串或对象之前、之后还是出现在同一位置。下面的示例使用 CompareTo()方法来比较 MyString 对象和 OtherString 对象。

```
string MyString = "Hello World";
string OtherString = "Hello World!";
int MyInt = MyString.CompareTo(OtherString);
Console.WriteLine( MyInt );                                    //向控制台显示 -1
```

3) CompareOrdinal()方法

CompareOrdinal()方法比较两个字符串对象而不考虑本地区域性。此方法将整个字

符串每5个字符分成一组，然后逐个比较，找到第一个不相同的ASCII码后退出比较，并求出两者ASCII的差。下面的示例使用CompareOrdinal()方法来比较两个字符串的值。

```
string MyString = "Hello World!";
Console.WriteLine(String.CompareOrdinal(MyString, "hello world!"));  //向控制台显示 -32
```

4) Equals()方法

Equals()方法能够轻松确定两个字符串是否相等。这种区分大小写的方法返回true或false布尔值。它可以在现有类中使用，如下面示例所示。以下示例使用Equals()方法来确定一个字符串对象是否包含短语"Hello World"。

```
string MyString = "Hello World";
Console.WriteLine(MyString.Equals("Hello World"));          //此示例向控制台显示 true
```

此方法还可作为静态方法使用。以下示例使用静态方法比较两个字符串对象。

```
string MyString = "Hello World";
string YourString = "Hello World";
Console.WriteLine(String.Equals(MyString, YourString));     //向控制台显示 true
```

【指点迷津】 在程序开发过程中，常常要判断字符串是否为空。在C#中，通常有4种判断字符串是否为空的方法：str.Length==0、str==string.Empty、str==""以及string.IsNullOrEmpty(str)。在这4种方法中，使用str.Length==0效率是最高的，但容易产生异常。string.IsNullOrEmpty(str)比str==string.Empty稍快。

2. 查找字符串

若要在一个字符串中搜索另一个字符串，可以使用IndexOf()。如果未找到搜索字符串，IndexOf()返回－1；否则，返回它出现的第一个位置的索引(从零开始)。具体格式如下：

```
public int IndexOf(char value)
public int IndexOf(string value)
public int IndexOf(char value,int startIndex)
public int IndexOf(string value,int startIndex)
public int IndexOf(char value, int startIndex,int count)
public intIndexOf(string value,int start,int count)
```

以下示例使用IndexOf()方法搜索字符'l'在字符串中的第一个匹配项。

```
string MyString = "Hello World";
Console.WriteLine(MyString.IndexOf('l'));                   //向控制台显示 2
```

LastIndexOf()方法类似于IndexOf()方法，但它返回特定字符在字符串中的最后一个匹配项的位置。它不区分大小写，并且使用从零开始的索引。以下示例使用LastIndexOf()方法搜索字符'l'在字符串中的最后一个匹配项。

```
string MyString = "Hello World";
Console.WriteLine(MyString.LastIndexOf('l'));               //向控制台输出 9
```

3. 格式化字符串

格式化字符串是内容可以在运行时动态确定的一种字符串。使用String类的静态方

法 Format()并在大括号中嵌入占位符来格式化字符串，这些占位符将在运行时替换为其他值。Format()方法的基本语法如下：

```
public static string Format(string format,object obj);
```

其中，format 用来指定字符串要格式化的形式，由零个或多个固定文本段与一个或多个格式项混合组成，其中索引占位符称为格式项，对应于列表中的对象。obj 代表要格式化的对象列表。

下面的示例说明了格式字符串的用法：

```
//C2 表示货币,其中 2 表示小数点后位数
Console.WriteLine(string.Format("{0:C2}",2));
//D2 表示十进制位数,其中 2 表示位数,不足用 0 占位
Console.WriteLine(string.Format("{0:D2}",2));
//E3 表示科学记数法,其中 3 表示小数点后保留的位数
Console.WriteLine(string.Format("{0:E3}",22233333220000));
//N 表示用分号隔开的数字
Console.WriteLine(string.Format("{0:N}",2340000));
//X 表示十六进制
Console.WriteLine(string.Format("{0:X}",12345));
//常规输出
Console.WriteLine(string.Format("{0:G}",12));
//按提供的格式(000.00→012.00,0.0→12.0)格式化的形式输出
Console.WriteLine(string.Format("{0:000.00}",12));
//F 表示浮点型,其中 3 表示小数点位数
Console.WriteLine(string.Format("{0:F3}",12));
```

4. 截取字符串

截取字符串需要使用 String 类的 Substring()方法，该方法从原始字符串中指定位置截取指定长度的字符，返回一个新的字符串。该方法的基本语法如下：

```
Substring(int startindex, int length)
```

其中，参数 startindex 索引从 0 开始，且最大值必须小于原字符串的长度，否则会编译异常；参数 length 的值必须不大于原字符串索引指定位置开始之后的字符串字符总长度，否则会出现异常。例如：

```
string s4 = "VisualC#Express";
System.Console.WriteLine(s4.Substring(6,2)) ;          //outputs  "C#"
```

下面的示例演示了如何从文件全名中获取文件的路径和文件名。

```
string strAllPath = "D:\\DataFiles\\Test.mdb";          //定义一个字符串,存储文件全名
string strPath = strAllPath.Substring(0, strAllPath.LastIndexOf("\\") + 1);  //获取文件路径
string strPath = strAllPath.Substring(strAllPath.LastIndexOf("\\") + 1);     //获取文件名
```

5. 拆分字符串

String 类的 Split()方法用于拆分字符串，此方法的返回值是包含所有拆分子字符串的数组对象，可以通过数组取得所有分隔的子字符串，其基本语法如下：

```
public string[] split (params char[] separator)
```

下面的代码示例演示如何使用 Split()方法分析字符串。作为输入，Split()采用一个字符数组指示哪些字符被用作分隔符。本示例中使用了空格、逗号、句点、冒号和制表符。一个含有这些分隔符的数组被传递给 Split()，并使用结果字符串数组分别显示句子中的每个单词。

```
char[] delimiterChars = { ' ', ',', '.', ':', '\t' };
string text = "one\ttwo three:four,five six seven";
System.Console.WriteLine("Original text: '{0}'", text);
string[] words = text.Split(delimiterChars);
System.Console.WriteLine("{0} words in text:", words.Length);
```

6. 连接字符串

连接是将一个字符串追加到另一个字符串末尾的过程。若要串联字符串变量，可以使用＋或＋＝运算符，也可以使用 Concat()、Join()或 Append()方法。“＋”运算符容易使用，且有利于提高代码的直观性。

注意，在字符串串联操作中，C＃编译器对 null 字符串和空字符串进行相同的处理，但它不转换原始 null 字符串的值。

1) Concat()方法

Concat()方法用于连接两个或多个字符串。Concat()方法也有多种重载形式，最常用的格式如下：

```
public static string Concat(params string[] values);
```

其中，参数 values 用于指定所要连接的多个字符串。例如：

```
newStr = String.Concat(strA," ",strB);
Console.WriteLine(newStr);       //"Hello World"
```

2) Join()方法

Join()方法利用一个字符数组和一个分隔符串构造新的字符串，常用于把多个字符串连接在一起，并用一个特殊的符号来分隔开。Join()方法的常用形式如下：

```
public static string Join(string separator,string[] values);
```

其中，参数 separator 为指定的分隔符，values 用于指定所要连接的多个字符串数组，下例用“^^”分隔符把"Hello"和"World"连起来。

```
newStr = "";
String[] strArr = {strA,strB};
newStr = String.Join("^^",strArr);
Console.WriteLine(newStr);       //"Hello^^World"
```

3) 连接运算符“＋”

String 支持连接运算符“＋”，可以方便地连接多个字符串，例如，下例把"Hello"和"World"连接起来。

```
newStr = "";
newStr = strA + strB;
Console.WriteLine(newStr);       //"HelloWorld"
```

7. 插入和填充字符串

String 类包含了在一个字符串中插入新元素的方法，可以用 Insert()方法在任意位置插入任意字符。Insert()方法用于在一个字符串的指定位置插入另一个字符串，从而构造一个新的字符串。Insert()方法也有多种重载形式，最常用的形式如下：

```
public string Insert(int startIndex,string value);
```

其中，参数 startIndex 用于指定所要插入的位置，从 0 开始索引；value 指定所要插入的字符串。下面的示例，在"Hello"的字符"H"后面插入"World"，构造一个串"HWorldello"。

```
newStr = "";
newStr = strA. Insert(1,strB);
Console. WriteLine(newStr);        //"HWorldello"
```

8. 删除和剪切字符串

1) Remove()方法

Remove()方法从一个字符串的指定位置开始删除指定数量的字符。最常用的形式为：

```
public string Remove(int startIndex,int count);
```

其中，参数 startIndex 用于指定开始删除的位置，从 0 开始索引；count 指定删除的字符数量。下例中，把"Hello"中的"ell"删掉。

```
newStr = strA. Remove(1,3);      Console. WriteLine(newStr);     //"Ho"
```

2) Trim()方法

若想把一个字符串首尾处的一些特殊字符剪切掉，如去掉一个字符串首尾的空格等，可以使用 String 的 Trim()方法。其形式如下：

```
public string Trim();
public string Trim(params char[] trimChars);
```

其中，参数 trimChars 数组包含了指定要去掉的字符，如果默认，则删除空格符号。下例中，实现了对“@Hello# $”的净化，去掉首尾的特殊符号。

```
newStr = "";
char[] trimChars = {'@','#','$',' '};
String strC = "@Hello#  $ ";
newStr = strC. Trim(trimChars);
Console. WriteLine(newStr);        //"Hello"
```

9. 复制字符串

String 类包括了复制字符串方法 Copy()和 CopyTo()，可以完成对一个字符串及其一部分的复制操作。

1) Copy()方法

若想把一个字符串复制到另一个字符数组中，可以使用 String 的静态方法 Copy()来实现，其形式为：

```
public string Copy(string str);
```

其中，参数 str 为需要复制的源字符串。该方法返回目标字符串。

2）CopyTo()方法

CopyTo()方法可以实现 Copy()同样的功能，但功能更为丰富，可以复制字符串的一部分到一个字符数组中。另外，CopyTo()不是静态方法，其形式为：

```
public void CopyTo(int sourceIndex,char[] destination,int destinationIndex,int count);
```

其中，参数 sourceIndex 为需要复制的字符起始位置，destination 为目标字符数组，destinationIndex 指定目标数组中的开始存放位置，而 count 指定要复制的字符个数。

下例中，把 strA 字符串"Hello"中的"ell"复制到字符数组 newCharArr 中，并在 newCharArr 中从第 2 个元素开始存放。

```
char[] newCharArr = new char[100];
strA.CopyTo(2,newCharArr,0,3);
Console.WriteLine(newCharArr);   //"Hel"
```

10. 替换字符串

要替换一个字符串中的某些特定字符或者某个子串，可以使用 Replace()方法来实现，其形式为：

```
public string Replace(char oldChar, char newChar);
public string Replace(string oldValue, string newValue);
```

其中，参数 oldChar 和 oldValue 为待替换的字符和子串，而 newChar 和 newValue 为替换后的新字符和新子串。下例把"Hello"通过替换变为"Hero"。

```
newStr = strA.Replace("ll","r");
Console.WriteLine(newStr);
```

由于字符串是不可变的，因此一个字符串对象一旦创建，值就不能再更改（在不使用不安全代码的情况下）。在使用其方法（如插入、删除操作）时，都要在内在中创建一个新的 String 对象，而不是在原对象的基础上进行修改，这就需要开辟新的内存空间。如果需要经常进行串修改操作，使用 String 类无疑是非常耗费资源的，这时需要使用 StringBuilder 类。

3.2.3 可变字符串

与 String 类相比，System.Text.StringBuilder 类可以实现动态字符串。此外，动态的含义是指在修改字符串时，系统不需要创建新的对象，不会重复开辟新的内存空间，而是直接在原 StringBuilder 对象的基础上进行修改。StringBuilder 类不像 String 类那样支持非常多的方法，在 StringBuilder 类上可以进行的处理仅限于替换、追加或删除字符串中的文本。

1. 创建可变字符串

StringBuilder 类位于命名空间 System.Text 中，使用时，可以在文件头通过 using 语句引入该空间。创建一个可变字符串 StringBuilder 对象需要使用 new 关键字，并可以对其进行初始化。下面的语句声明了一个 StringBuilder 对象 myStringBuilder，并初始化为"Hello"：

```
StringBuilder myStringBuilder = new StringBuilder("Hello");
```

如果不使用 using 关键字在文件头引入 System. Text 命名空间，也可以通过空间限定来声明 StringBuilder 对象：

```
System.Text.StringBuilder  myStringBuilder = new StringBuilder("Hello");
```

在声明时，也可以不给出初始值，然后通过其方法进行赋值。

2. 设置可变字符串容量

StringBuilder 对象为动态字符串，可以对其设置好的字符数量进行扩展。另外，还可以设置一个最大长度，这个最大长度称为该 StringBuilder 对象的容量（Capacity）。为 StringBuilder 设置容量的意义在于，当修改 StringBuilder 字符串时，当其实际字符长度（即字符串已有的字符数量）未达到其容量之前，StringBuilder 不会重新分配空间；当达到容量时，StringBuilder 会在原空间的基础上，自动不进行设置，StringBuilder 默认初始分配 16 个字符长度。有两种方式来设置一个 StringBuilder 对象的容量。

1）使用构造函数

StringBuilder 构造函数可以接受容量参数，例如，下面声明一个 StringBuilder 对象 sb2，并设置其容量为 100。

```
//使用构造函数
StringBuilder sb2 = new StringBuilder("Hello",100);
```

2）使用 Capacity 读/写属性

Capacity 属性指定 StringBuilder 对象的容量，例如下面语句首先创建一个 StringBuilder 对象 sb3，然后利用 Capacity 属性设置其容量为 100。

```
//使用 Capacity 属性
StringBuilder sb3 = new StringBuilder("Hello");
sb3.Capacity = 100;
```

3. 追加操作

追加一个 StringBuilder 是指将新的字符串添加到当前 StringBuilder 字符串的结尾处，可以使用 Append()和 AppendFormat()方法来实现这个功能。

1）Append()方法

Append()方法实现简单的追加功能，其常用形式为：

```
public StringBuilder Append(object value);
```

其中，参数 value 既可以是字符串类型，也可以是其他的数据类型，如 bool、byte、int 等。下例中，把一个 StringBuilder 字符串"Hello"追加为"Hello World!"。

```
//Append()
StringBuilder sb4 = new StringBuilder("Hello");
sb4.Append(" World!");
```

2）AppendFormat()方法

AppendFormat()方法可以实现对追加部分字符串的格式化，还可以定义变量的格式，并将格式化后的字符串追加在 StringBuilder 后面。其常用的形式为：

```
StringBuilder AppendFormat(string format,params object[] args);
```

其中,args 数组指定所要追加的多个变量。format 参数包含格式规范的字符串,其中包括一系列用大括号括起来的格式字符,如{0:u}。这里,0 代表对应 args 参数数组中的第 0 个变量,而 u 定义其格式。下例中,把一个 StringBuilder 字符串"Today is"追加"Today is 当前日期"。

```
//AppendFormat
StringBuilder sb5 = new StringBuilder("Today is ");
sb5.AppendFormat("{0:yyyy-MM-dd}",System.DateTime.Now);
Console.WriteLine(sb5);          //形如: "Today is 2008-10-20"
```

4. 插入操作

StringBuilder 的插入操作是指将新的字符串插入到当前的 StringBuilder 字符串的指定位置,如"Hello"变为"Heeeello"。可以使用 StringBuilder()类的 Insert()方法来实现这个功能,其常用形式为:

```
public StringBuilder Insert(int index, object value);
```

其中,参数 index 指定所要插入的位置,并从 0 开始索引,如 index=1,则会在原字符串的第 2 个字符之前进行插入操作; 同 Append()一样,参数 value 并不仅仅是只可取字符串类型。

5. 修改操作

可以使用 StringBuilder 类的 Remove()方法从当前字符串中删除字符,也可以使用 Replace()方法在当前字符串中用一个字符或者子字符串全部替换另一个字符或者字符串。

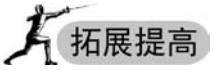
下面的代码演示了 StringBuilder 类的用法。该代码将当前字符串中的每个字符的 ASCII 值加 1,形成非常简单的加密模式。详细代码如下所示:

```
StringBuilder greetingBuilder = new StringBuilder("Hello from all the guys at Xuchang University,"150);
greetingBuilder.AppengFormat("We do hope you enjoy this book as much as we enjoyed writing it.");

for(int i='z'; i>'a'; i--)
{
    char old1 = (char)i;
    char new1 =(char)(i+1);
    greetingBuilder = greetingBuilder.Replace(old1,new1);
}

for(int i='Z'; i>'A'; i--)
{
    char old1 = (char)i;
    char new1 =(char)(i+1);
    greetingBuilder = greetingBuilder.Replace(old1,new1);
}
```

这段代码也可以使用 String.Replace()方法完成,但是由于 Replace()需要分配一个新字符串,整个加密过程需要在堆上有一个能存储大约 2800 个字符的字符串对象,该对象最

终等待被垃圾回收机制回收。在上述代码中，为存储字符串而分配的总的存储单元是用于StringBuilder实例的150个字符。

通过上面的介绍，可以看出StringBuilder与String在许多操作（如插入、删除、替换）上是非常相似的。在操作性能和内存效率方面，StringBuilder要比String好得多，可以避免产生太多的临时字符串对象，特别是对于经常重复进行修改的情况更是如此。另外，String类提供了更多的方法，可以使开发能够更快地实现应用。在两者的选择上，一般而言，使用StringBuilder类执行字符串的任何操作，而使用String类存储字符串或显示最终结果。



1. 正则表达式

在编写字符串的处理程序时，经常会有查找符合某些复杂规则的字符串的需要。正则表达式就是用于描述这些规则的工具。换句话说，正则表达式就是记录文本规则的代码。目前为止，许多编程语言和工具都包含对正则表达式的支持，C#也不例外，C#基础类库中包含有一个命名空间（System.Text.RegularExpressions）和一系列可以充分发挥规则表达式威力的类（Regex、Match、Group等）。借助网络资源，了解C#正则表达式的用法，可以提高程序设计技能。

2. 编码

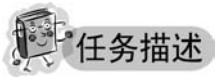
众所周知，计算机只能识别二进制数字，如1010、1001。我们在屏幕所看到的文字，字符都是进行二进制转换后的结果。将文字按照某种规则转换为二进制存储在计算机上，这一个过程叫字符编码（Encoding），反之就是字符解码。目前存在多种字符编码方式，一组二进制数字根据不同的解码方式，会得到不同的结果，有时甚至会得到乱码。这也就是为什么我们打开网页时有时会是乱码，打开一个文本文件有时也是乱码，而换了一种编码就恢复正常了。CLR中的所有字符都是用16位Unicode来表示的。CLR中的Encoding就是用于字节和字符之间的转换的。

CLR中的Encoding是在System.Text命名空间下的，它是一个抽象类，所以不能被直接实例化，它主要有如下派生类：ASCIIEncoding、UnicodeEncoding、UTF32Encoding、UTF7Encoding和UTF8Encoding。可以根据需要选择一个合适的Encoding来进行编码和解码；也可以调用Encoding的静态属性ASCII、Unicode、UTF32、UTF7、UTF8来构造一个Encoding。其中Unicode是表示16位Encoding。调用静态属性和实例化一个子类的效果是一样的，如下代码。

```
Encoding encodingUTF8 = Encoding.UTF8;
Encoding encodingUTF8 = new UTF8Encoding(true);
```


视频讲解

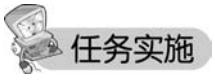
3.3 集　　合



在程序开发中，虽然可以使用数组来管理一组具有相同数据类型的数据，但是，数组的

大小是事先确定的，不便于数据的动态编辑。例如，在学生成绩管理系统中，使用数组存储学生信息时，必须事先指定学生的人数，而在实际的系统，每个班的学生人数是不相同的，为了实现数据的动态添加，就需要使用集合来管理数据。本任务使用集合来进一步优化学生成绩管理系统的数据管理功能，如图 3-5 所示。

图 3-5　学生列表



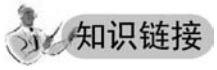
（1）新建项目。启动 Visual Studio 2017，新建控制台项目 ImprovetStudent。

（2）修改代码。项目初始化以后，在主窗口显示的 Program.cs 文件中添加如下代码行：

```
//学生结构体定义
public struct Student
{
    public string id;
    public string name;
    public string grade1;
    public string grade2;
    public string grade3;
    public int total;
    public int average;
}
//主函数
static void Main(string[] args)
{
    const int NUM = 3;              //学生人数
    Student stu;                    //声明学生结构
    //建立学生列表
    List<Student> listStudent = new List<Student>();
    //输入学生信息
    for (int i = 0; i < NUM; i++)
```

```
    {
        Console.Write("输入第{0}个学生学号: ", i + 1);
        stu.id = Console.ReadLine();
        Console.Write("输入第{0}个学生姓名: ", i + 1);
        stu.name = Console.ReadLine();
        Console.Write("输入第{0}个学生语文成绩: ", i + 1);
        stu.grade1 = Console.ReadLine();
        Console.Write("输入第{0}个学生数学成绩: ", i + 1);
        stu.grade2 = Console.ReadLine();
        Console.Write("输入第{0}个学生英语成绩: ", i + 1);
        stu.grade3 = Console.ReadLine();
        //计算总分和平均分
        stu.total = Convert.ToInt32(stu.grade1) + Convert.ToInt32(stu.grade2) + Convert.
        ToInt32(stu.grade3);
        stu.average = stu.total / 3;
        listStudent.Add(stu);   //添加学生到学生列表
    }
    //输出学生成绩
    Console.WriteLine("                    学生成绩单");
    Console.WriteLine("|-----------------------------------------|");
    Console.WriteLine("| 学  号 | 姓名 |语文|数学|英语|总 分|平均分|");
    Console.WriteLine("|-----------------------------------------|");
    //遍历学生列表
foreach (var s in listStudent)
    {
        Console.WriteLine("|{0,8}|{1,3}|{2,4}|{3,4}|{4,4}|{5,5}|{6,6:f2}|", s.id,
                          s.name, s.grade1, s.grade2, s.grade3, s.total, s.average);
        Console.WriteLine("|-----------------------------------------|");
    }
}
```

(3) 编译和运行程序。按 Ctrl+F5 组合键运行该程序,运行结果如图 3-5 所示。



3.3.1 集合基础

对于很多应用程序,需要创建和管理相关对象组。有两种方式可以将对象分组:创建对象数组以及创建对象集合。数组对于创建和处理固定数量的强类型对象最有用。集合提供一种更灵活的处理对象组的方法。与数组不同,处理的对象组可根据程序更改的需要动态地增长和收缩。

集合是类,因此必须声明新集合后,才能向该集合中添加元素。许多常见的集合是由.NET Framework 提供的,每一类型的集合都是为特定用途设计的。可以通过使用 System.Collections.Generic 命名空间中的类来创建泛型集合。在.NET Framework 2.0 之前,不存在泛型。现在泛型集合类通常是集合的首选类型。泛型集合类是类型安全的,如果使用值类型,是不需要装箱操作的。如果要在集合中添加不同类型的对象,且这些对象不是相互派生的,例如在集合中添加 int 和 string 对象,就只需基于对象的集合类。

表 3-2 列出了 System. Collections. Generic 命名空间中一些常用集合类的功能。

表 3-2　System. Collections. Generic 命名空间中一些常用集合类的功能

类	描　　述
Dictionary < TKey, TValue >	表示根据键进行组织的键/值对的集合
List < T >	表示可通过索引访问的对象的列表。提供用于对列表进行搜索、排序和修改的方法
Queue < T >	表示对象的先进先出（FIFO）集合
SortedList < TKey, TValue >	表示根据键进行排序的键/值对的集合，而键基于的是相关的 IComparer < T > 实现
Stack < T >	表示对象的后进先出（LIFO）集合

在.NET Framework 4 中，System. Collections. Concurrent 命名空间中的集合可提供有效的线程安全操作，以便从多个线程访问集合项。当有多个线程并发访问集合时，应使用 System. Collections. Concurrent 命名空间中的类代替 System. Collections. Generic 和 System. Collections 命名空间中的对应类型。

System. Collections 命名空间中的类不会将元素存储为指定类型的对象，而是存储为 object 类型的对象。只要有可能，就应使用 System. Collections. Generic 或 System. Collections. Concurrent 命名空间中的泛型集合来替代 System. Collections 命名空间中的旧类型。

表 3-3 列出了一些 System. Collections 命名空间中常用的集合类的用法。

表 3-3　System. Collections 命名空间中常用的集合类的用法

类	描　　述
ArrayList	表示大小根据需要动态增加的对象数组
Hashtable	表示根据键的哈希代码进行组织的键/值对的集合
Queue	表示对象的先进先出（FIFO）集合
Stack	表示对象的后进先出（LIFO）集合

3.3.2　列表

.NET Framework 为动态列表提供了类 ArrayList 和 List。ArrayList 代表了可被单独索引的对象的有序集合。它基本上可以替代一个数组。但是，与数组不同的是，可以使用索引在指定的位置添加和移除项目，动态数组会自动重新调整它的大小。它也允许在列表中进行动态内存分配、增加、搜索各项并排序。System. Collections. Generic 命名空间中的类 List 的用法非常类似于 System. Collections 命名空间中的 ArrayList 类，这个类实现了 IList、ICollection 和 IEnumerable 接口。

对于新的应用程序，通常可以使用泛型类 List < T >替代非泛型类 ArrayList，而且 ArrayList 类的方法与 List < T >非常相似，所以本节将只讨论如何使用 List < T >类。

1. 创建列表

调用默认的构造函数就可以创建对象列表。在泛型类 List < T >中，必须在声明中为列表的值指定类型。下面的代码说明了如何声明一个包含 int 和 string 的列表。

```
List<int> intList = new List<int>();
List<string> strList = new List<string>();
```

使用默认的构造函数创建一个空列表。元素添加到列表中后，列表的容量就会扩大为可接纳 4 个元素。如果添加了第 5 个元素，列表的大小就重新设置为包含 8 个元素。如果 8 个元素还不够，列表的大小就重新设置为 16，每次都会将列表的容量重新设置为原来的 2 倍。为节省时间，如果事先知道列表中元素的个数，就可以用构造函数定义其容量。下面的代码创建一个容量为 10 个元素的集合。如果该容量不足以容纳要添加的元素，就把集合的大小重新设置为 20 或者 40，每次都是原来的 2 倍。

```
List<int> intList = new List<int>(10);
```

使用 Capacity 属性可以获取和设置集合的容量。

```
intList.Capacity = 20;
```

容量与集合中元素的个数不同。集合中元素的个数可以用 Count 属性读取。当然，容量总是大于或等于元素个数。只要不把元素添加到列表中，元素个数就是 0。

```
Console.WriteLine(intList.Count);
```

如果已经将元素添加到列表中，且不希望添加更多的元素，就可以调用 TrimExcess() 方法，去除不需要的容量。但是，重新定位是需要时间的，所以如果元素个数超过了容量的 90%，TrimExcess()方法将什么也不做。

```
int List.TrimExcess();
```

2. 添加元素

使用 Add()方法可以给列表添加元素。Add()方法将对象添加到列表的结尾处，例如：

```
List<int> intList = new List<int>();
intList.Add(1);
intList.Add(2);
List<string> strList = new List<string>();
strList.Add("one");
strList.Add("two");
```

使用 List<T>类的 AddRange()方法可以一次给集合添加多个元素。AddRange()方法的参数是 IEnumerable<T>类型对象，所以也可以传送一个数组，例如：

```
strList.AddRange(new string[ ]{"one","two","three"});
```

3. 插入元素

使用 Insert()方法可以在列表的指定位置插入元素，位置从 0 开始索引。例如：

```
intList.Insert(3,6);
```

方法 InsertRange()提供了插入大量元素的容量，类似于前面的 AddRange()方法。如果索引集大于集合中的元素个数，就抛出 ArgumentOutOfRangeException 类型的异常。

4. 访问元素

执行了 IList 和 IList 接口的所有类都提供了一个索引器，所以可以使用索引器，通过

传送元素号来访问元素。第一个元素可以用索引值 0 来访问。例如指定 intList[3],可以访问列表 intList 中的第 4 个元素:

```
int num = intList[3];
```

可以用 Count 属性确定元素个数,再使用 for 循环语句迭代集合中的每个元素,使用索引器访问每一项,例如:

```
for (int i = 0; i < intList.Count; i++)
{
    Console.WriteLine(intList[i]);
}
```

List 执行了接口 IEnumerable,所以也可以使用 foreach 语句迭代集合中的元素。编译器解析 foreach 语句时,利用了接口 IEnumerable 和 IEnumerator。

```
foreach (int i in intList)
{
    Console.WriteLine(i);
}
```

5. 删除元素

删除元素时,可以利用索引或传送要删除的元素。下面的代码把 3 传送给 RemoveAt(),删除第 4 个元素:

```
intList. RemoveAt(3);
```

也可以直接把对象传送给 Remove()方法,删除这个元素。例如:

```
intList.Remove(3);                    //删除列表中元素 3
```

方法 RemoveRange()可以从集合中删除许多元素。它的第一个参数指定了开始删除的元素索引,第二个参数指定了要删除的元素个数。

```
int index = 3;
int count = 5;
intList. RemoveRange (index, count);
```

要删除集合中的所有元素,可以使用 ICollection<T>接口定义的 Clear()方法。

6. 搜索

有不同的方式在集合中搜索元素。可以获得要查找的元素的索引,或者搜索元素本身。可以使用的方法有 IndexOf()、LastIndexOf()、FindIndex()、FindLastIndex()、Find()和 FindLast()。如果只检查元素是否存在,List<T>类提供了 Exists()方法。

方法 IndexOf()需要将一个对象作为参数,如果在集合中找到该元素,这个方法就返回该元素的索引。如果没有找到该元素,就返回-1。IndexOf()方法使用 IEquatable 接口来比较元素。例如:

```
int index = intList. IndexOf(3);
```

使用方法 IndexOf(),还可以指定不需要搜索整个集合,但必须指定从哪个索引开始搜

索以及要搜索的元素个数。

3.3.3 队列

队列是其元素以先进先出(FIFO)的方式来处理的集合。先放在队列中的元素会先读取。队列的例子有在机场排的队、人力资源部中等待处理求职信的队列、打印队列中等待处理的打印任务、以循环方式等待 CPU 处理的线程。另外,还常常有元素根据其优先级来处理的队列。例如,在机场的队列中,商务舱乘客的处理要优先于经济舱的乘客。这里可以使用多个队列,一个队列对应一个优先级。在机场,这是很常见的,因为商务舱乘客和经济舱乘客有不同的登记队列。打印队列和线程也是这样。可以为一组队列建立一个数组,数组中的一项代表一个优先级。在每个数组项中,都有一个队列,其处理按照 FIFO 的方式进行。

在.NET 的 System. Collections 命名空间中有非泛型类 Queue,在 System. Collections. Generic 命名空间中有泛型类 Queue<T>。这两个类的功能非常类似,但泛型类是强类型化的,定义了类型 T,而非泛型类基于 Object 类型。

在内部,Queue<T>类使用 T 类型的数组,这类似于 List<T>类型。另一个类似之处是它们都执行 ICollection 和 IEnumerable 接口。Queue 类执行 ICollection、IEnumerable 和 ICloneable 接口。Queue<T>类执行 IEnumerable 和 ICloneable 接口。Queue<T>泛型类没有执行泛型接口 ICollection<T>,因为这个接口用 Add()和 Remove()方法定义了在集合中添加和删除元素的方法。

队列与列表的主要区别是队列没有执行 IList 接口。所以不能用索引器访问队列。队列只允许添加元素,该元素会放在队列的尾部(使用 Enqueue()方法),从队列的头部获取元素(使用 Dequeue()方法)。

图 3-6 显示了队列的元素。Enqueue()方法在队列的一端添加元素,Dequeue()方法在队列的另一端读取和删除元素。用 Dequeue()方法读取元素,将同时从队列中删除该元素。再调用一次 Dequeue()方法,会删除队列中的下一项。

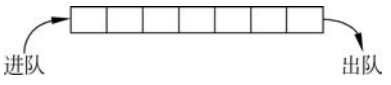


图 3-6 队列

Queue 和 Queue<T>类的方法如表 3-4 所示。

表 3-4 队列常用方法

方 法	说 明
Enqueue()	在队列一端添加一个元素
Dequeue()	在队列的头部读取和删除一个元素。如果在调用 Dequeue()方法时,队列中不再有元素,就抛出 InvalidOperationException 异常
Peek()	在队列的头部读取一个元素,但不删除它
Count	返回队列中的元素个数
TrimExcess()	重新设置队列的容量。Dequeue()方法从队列中删除元素,但不会重新设置队列的容量。要从队列的头部去除空元素,应使用 TrimExcess()方法
Contains()	确定某个元素是否在队列中,如果是,就返回 true
CopyTo()	把元素从队列复制到一个已有的数组中
ToArray()	ToArray()方法返回一个包含队列元素的新数组

下面的代码演示了 Queue < T >类的基本用法。

```
Queue < string > numbers = new Queue < string >();          //实例化队列对象
//向队列中添加元素
numbers.Enqueue("one");
numbers.Enqueue("two");
numbers.Enqueue("three");
numbers.Enqueue("four");
numbers.Enqueue("five");
//遍历队列中的元素
foreach( string number in numbers )
{
    Console.Write("{0} ",number);
}
//调用队列方法
Console.WriteLine("\nDequeuing '{0}'", numbers.Dequeue());
Console.WriteLine("Peek at next item to dequeue: {0}", numbers.Peek());
Console.WriteLine("Dequeuing '{0}'", numbers.Dequeue());
```

上面代码会产生如下的输出结果：

```
one two three four five
Dequeuing 'one'
Peek at next item to dequeue: two
Dequeuing 'two'
```

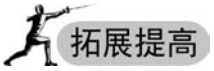
3.3.4 字典

字典(Dictionary)表示一种复杂的数据结构，这种数据结构允许按照某个键来访问元素。字典也称为映射或散列表。字典的主要特性是能根据键快速查找值，也可以自由添加和删除元素，这有点像 List < T >，但没有在内存中移动后续元素的性能开销。

Dictionary 泛型类提供了从一组键到一组值的映射。字典中的每个添加项都由一个值及其相关联的键组成。通过键来检索值的速度是非常快的，接近于 O(1)，这是因为 Dictionary 类是作为一个哈希表来实现的。在 C# 中，Dictionary 提供快速的基于键值的元素查找。当有很多元素时可以使用它。它需要引用的命名空间是 System. Collection. Generic。下面的代码演示了字典的基本用法：

```
//定义
Dictionary < string, string > openWith = new Dictionary < string, string >();
//添加元素
openWith.Add("txt", "notepad.exe");
openWith.Add("bmp", "paint.exe");
//取值
Console.WriteLine("For key = \"rtf\", value = {0}.", openWith["rtf"]);
//更改值
openWith["rtf"] = "winword.exe";
//查看
Console.WriteLine("For key = \"rtf\", value = {0}.", openWith["rtf"]);
//遍历 Key
foreach (var item in openWith.Keys)
```

```
{
    Console.WriteLine("Key = {0}", item);
}
//遍历 value
foreach (var item in openWith.Values)
{
    Console.WriteLine("value = {0}", item);
}
```



1. 泛型

泛型(Generic)是C# 2.0推出的新语法,不是语法糖,而是C# 2.0由框架升级提供的功能。在编程程序时,开发人员经常会遇到功能非常相似的模块,只是它们处理的数据不一样。但我们没有办法,只能分别写多个方法来处理不同的数据类型。这个时候,那么问题来了,有没有一种办法,用同一个方法来处理传入不同种类型参数的办法呢?泛型的出现就是专门来解决这个问题的。

泛型允许开发人员延迟编写类或方法中的编程元素的数据类型的规范,直到在程序中使用它时。换句话说,泛型允许编写一个可以与任何数据类型一起工作的类或方法。可以通过数据类型的替代参数编写类或方法的规范。当编译器遇到类的构造函数或方法的函数调用时,它会生成代码来处理指定的数据类型。

2. 使用集合初始值设定项初始化字典

Dictionary<TKey,TValue>包含键/值对集合。其Add()方法采用两个参数:一个用于键;另一个用于值。若要初始化Dictionary<TKey,TValue>,一种方法是将每组参数括在大括号中,另一种方法是使用索引初始值设定项。下面的代码示例中,使用类型Dictionary<TKey,TValue>的实例初始化StudentName。第一个初始化使用具有两个参数的Add()方法。编译器为每对Add键和int值生成对StudentName的调用。第二个初始化使用Dictionary类的公共读取/写入索引器方法:

```
var students = new Dictionary<int, StudentName>()
{
    { 111, new StudentName { FirstName = "Sachin", LastName = "Karnik", ID = 211 } },
    { 112, new StudentName { FirstName = "Dina", LastName = "Salimzianova", ID = 317 } },
    { 113, new StudentName { FirstName = "Andy", LastName = "Ruth", ID = 198 } }
};
var students2 = new Dictionary<int, StudentName>()
{
    [111] = new StudentName { FirstName = "Sachin", LastName = "Karnik", ID = 211 },
    [112] = new StudentName { FirstName = "Dina", LastName = "Salimzianova", ID = 317 } ,
    [113] = new StudentName { FirstName = "Andy", LastName = "Ruth", ID = 198 }
};
```

3.4 知识点提炼

(1) C#提供了能够存储多个相同类型变量的集合,这种集合就是数组。数组是同一数据类型的一组值,它属于引用类型。

(2) 数组在使用之前必须先定义。一个数组的定义必须包含元素类型、数组维数和每个维数的上下限。

(3) 数组在使用之前必须进行初始化。初始化数组有两种方法：动态初始化和静态初始化。动态初始化需要借助 new 运算符，为数组元素分配内存空间，并为数组元素赋初值。静态初始化数组时，必须与数组定义结合在一起，否则会出错。

(4) C#中的字符包括数字字符、英文字母、表达符号等。C#提供的字符类型按照国际上公认的标准，采用 Unicode 字符集。要得到字符的类型，可以使用 System.Char 命名空间中的内置静态方法。

(5) C#语言中，string 类型是引用类型，其表示零或更多个 Unicode 字符组成的序列。字符串常用的属性有 Length、Chars 等。利用字符串类的方法可以实现对字符串的处理操作。

(6) 可变字符串类 StringBuilder 创建了一个字符串缓冲区，允许重新分配个别字符，这些字符是内置字符串数据类型所不支持的。

(7) 集合提供一种动态对数据分组的方法。.NET Framework 提供了泛型和非泛型两大集合类型。

3.5 思考与练习

1. 下列关于数组访问的描述中，(　　)是不正确的。
   A. 数组元素索引是从 0 开始的
   B. 对数组元素的所有访问都要进行边界检查
   C. 如果使用的索引小于 0 或大于数组的大小，编译器将抛出一个 IndexOutOfRangeException 异常
   D. 数组元素的访问从 1 开始，到 Length 结束
2. 数组 pins 的定义如下：

```
int[] pins = new int[4]{9,2,3,1};
```

   则 pins[1] =(　　)。
   A. 1　　B. 2　　C. 3　　D. 9
3. 有说明语句“double[,] tab=new double[2,3];”，那么下面叙述正确的是(　　)。
   A. tab 是一个数组维数不确定的数组，使用时可以任意调整
   B. tab 是一个有两个元素的一维数组，它的元素初始值分别是 2,3
   C. tab 是一个二维数组，它的元素个数一共有 6 个
   D. tab 是一个不规则数组，数组元素的个数可以变化
4. 下列关于数组的描述中，(　　)是不正确的。
   A. String 类中的许多方法都能用在数组中
   B. System.Array 类是所有数组的基类
   C. String 类本身可以被看作是一个 System.Char 对象的数组
   D. 数组可以用来处理数据类型不同的批量数据

5. 下面代码实现数组 array 的冒泡排序，画线处应填入(　　)。

```
int[ ] array = { 20, 56, 38, 45 };
int temp;
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < ______; j++)
    {
        if (a[j] < a[j + 1])
        {
            temp = a[j];
            array[j] = a[j + 1];
            array[j + 1] = temp;
        }
    }
}
```

A. 4－i　　B. i　　C. i＋1　　D. 3－i

6. 在 C＃中，将路径名“C:\Documents\”存入字符串变量 path 中的正确语句是(　　)。

A. path＝"C:\\Documents\\";　　B. path＝"C://Documents//";

C. path＝"C:\Documents\";　　D. path＝"C:\/Documents\/";

7. 从控制台输入班级人数，将每个人的年龄放入数组，计算所有人的年龄总和和平均年龄，并输出年龄最大的学生。

8. 编写一个控制台程序获取字符串中相同的字符及其个数。

9. 分拣奇偶数。将字符串"1 2 3 4 5 6 7 8 9 10"中的数据按照“奇数在前、偶数在后”的格式进行调整。