第1章　C# 编程基础

1.1　Visual Studio. NET 和 C#

C#（读做“C sharp”）是一种现代的面向对象的程序开发语言。它的语法简洁、与 Web 结合紧密、安全性完整、具有错误处理、版本处理技术等优点。其运行环境是由微软公司开发的 Visual Studio. NET（简称 VS）。用 VS 可以快速开发种类丰富的应用程序，如 Windows 桌面应用程序（Winforms）、Windows 服务程序（Windows Services）、Web 应用程序（ASP. NET）和 Web 服务程序（Web Services）。本书中使用的是 VS 2005 版本，它支持的是 . NET Framework 2. 0。

1.2　第一个 C#控制台应用程序

控制台应用程序是一种在后台运行的程序，没有独立的窗口，它不像 Windows 应用程序那样，可以用鼠标单击按钮等控件来实现操作，而是通过命令行来运行的。

【例 1-1】 创建一个 C#控制台程序，在屏幕上打印出一条欢迎信息，如图 1-1 所示。

实现步骤如下：

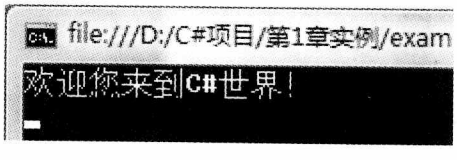

图 1-1　程序运行结果

1）首先启动 VS 2005，打开“开始”→“程序”→“Microsoft Visual Studio 2005”→“Microsoft Visual Studio 2005”选项，如图 1-2 所示。

图 1-2　启动 Visual Studio 2005

2）进入 VS. NET 2005 开发环境，打开“文件”→“新建”→“项目”选项，打开“新建项目”对话框，如图 1-3 所示。

3）在左窗格的项目类型中选择“Visual C#”，在右窗格的模板中选择“控制台程序”。

4）在“名称”文本框中输入项目名称“example1-1”，然后选择项目的保存位置，如

D:\C#项目\第 1 章实例。

5）单击“确定”按钮。这样我们就创建了一个 C#控制台应用程序项目，如图 1-4 所示。

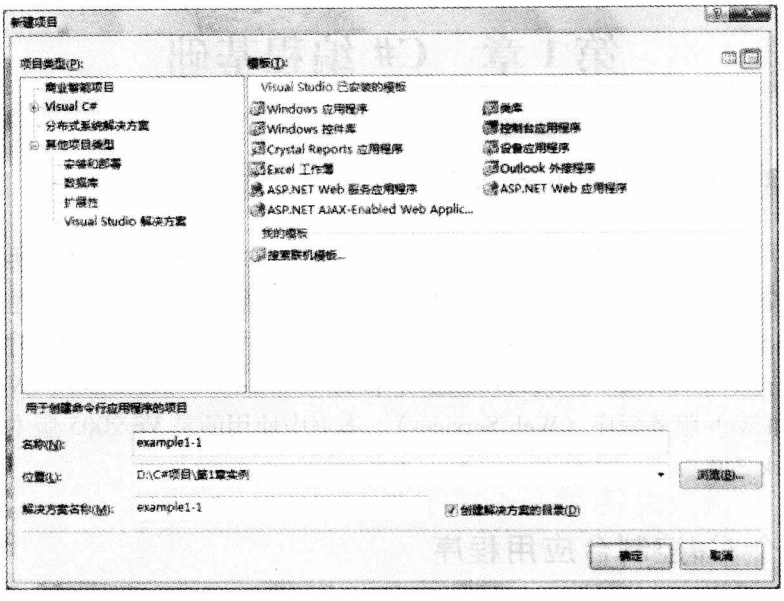

图 1-3 “新建项目”对话框

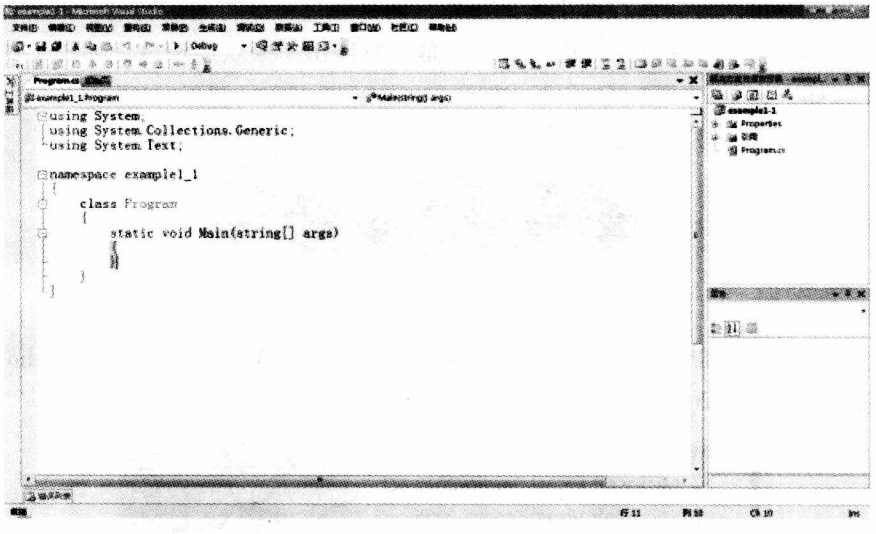

图 1-4 创建 C#控制台应用程序

6）在 Main()方法中添加如下代码：

```
Console. WriteLine("欢迎您来到 C#世界!");
Console. ReadLine();
```

7）打开“生成”→“生成 example1. 1”选项。如果在错误列表中显示“生成：1 成功或最新，0 失败，0 被跳过”，则表示代码编译正确，可以运行了！

8）单击“调试”→“启动调试”选项或按快捷键〈F5〉。如果出现图 1-1 所示的程序运行结果，那么，恭喜你已经成功运行出自己的第一个 C#程序了！

前面创建新项目 example1. 1 的时候，Visual Studio. NET 2005 就已经创建了一个与项目同名的 example1. 1 文件夹，叫做“解决方案”文件夹。“解决方案”包含一个或多个项目，以及定义整个解决方案所需的文件和元数据。每个项目可以解决一个独立的问题。本章中的任务都是在一个解决方案的一个项目中完成的。用户可以在“解决方案资源管理器”窗口中管理 example1. 1 项目中所包含的各类文件。在这个窗口中单击，“显示所有文件”按钮，如图 1-5 所示。

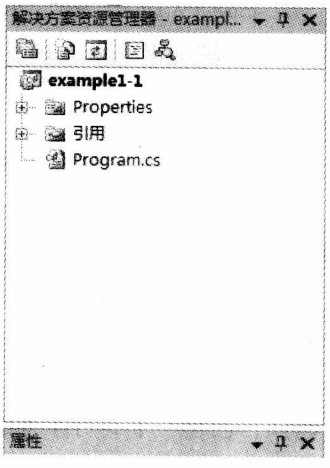

图 1-5 “解决方案资源管理器”窗口

这个文件夹中包含了 C#项目文件 example1. 1. csproj 和其他关联文件；每个新项目都创建了 bin、obj 和 Properties 3 个文件夹；bin 和 obj 这两个文件夹下都有一个 Debug 子目录，其中包含可执行文件 example1. 1. exe。现在只需要了解以下两个文件：

1）Program. cs：它是源程序文件，编写的代码就保存在这个文件中。从图 1-5 中可以看出，这个程序包含了 C#执行程序所必需的基本项，即一个命名空间和一个包含 Main( )方法的 Program 类，其中，Main( )方法是程序的入口点。关于命名空间，后面的章节会详细介绍。

2）example1. 1. exe：这个文件位于 bin \ Debug 文件夹下，是项目编译后的可执行文件，可以直接运行。

1.3 C#基本编程元素

下面介绍 C#语言的基本语法和结构。

1.3.1 数组

1. 一维数组

在一维数组中，一组相同类型的元素组合在一起，使用一个通用的名称，并用一个下标（索引）来访问数组中的元素，下标从 0 开始。

例如：Array1 = new int [5];

声明了一个由 5 个整型元素组成的数组 Array1，包含 Array [0] ～Array [4] 的 5 个元素，每个元素的取值都是 0。

```
int [ ] Array1 = new int [ ] {1,2,3,4,5,6};
```

定义了一个整型数组并完成初始化。用户也可以省略关键字 new，而使用下列方式：

```
int [ ] Array1 = {1,2,3,4,5,6};
```

【例 1-2】创建一个控制台应用程序，打印一维数组的每个元素，如图 1-6 所示。

实现步骤如下：

1）在 VS 中创建一个名为 example1-2 的控制台应用程序。

2）在 Main( )方法中添加如下代码：

```
string[ ] Array = {"Sun","Sat","Mon","Tue","Wed","Thu","Fri"};
for (int i = 0; i < Array. Length; i ++)
//用"数组名 . Length"获取数组长度
{
    Console. WriteLine(Array [i]);
}
Console. ReadLine( );
```

图 1-6　一维数组的应用示例

3）运行程序，观察结果。

2. 二维数组

二维数组访问其中元素时，要指出行索引和列索引。

例如，创建一个 3 行 4 列的二维数组。

```
int Array1 = new int [3,4] ;
```

数组 Array1 中包含了 Array1 [0][0]～Array1 [0][3]、Array1 [1][0]～Array1 [1][3] 和 Array1[2][0]～Array1[2][3]的共 12 个元素，每个元素的取值默认都是 0。我们可以用给定的值对数组元素进行初始化。

【例 1-3】创建一个控制台应用程序，用于演示二维数组的应用，如图 1-7 所示。

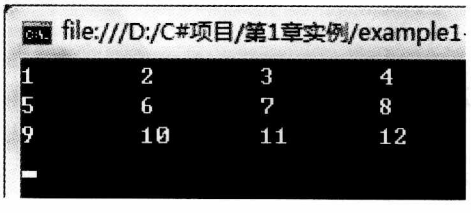

图 1-7　二维数组的应用示例

实现步骤如下：

1）在 VS 中创建一个名为 example1-3 的控制台应用程序。

2）在 Main( )方法中添加如下代码：

```
// 声明一个 3 行 4 列的二维数组,用于存放矩阵 A,并初始化矩阵 A
int[ , ] A = new int[3, 4] {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
int row = 3,colunm = 4;
for (int i = 0; i < row; i ++)
    {
    // 逐一打印各行中所有元素
    for (int j = 0; j < colunm; j ++)
    {
        Console. Write(A[i, j] + "\t");
    }
```

```
            Console. WriteLine( );
        }
        Console. ReadLine( );
```

3）运行程序，观察结果。

1.3.2 集合

System. Collections 命名空间包含了各种对象集的接口和类的定义。这些接口和类定义了各种对象（如列表、队列、位数组、哈希表和字典）的集合。集合是由 System. Collections 命名空间提供的一个容器，它是对一般数组功能的扩展，集合类的元素类型是 object。下面介绍两种常用的集合。

1. 列表（ArrayList）

ArrayList 类用于建立可变长数组，其容量会随着需要而适当地扩充，在该集合中可存放任何类型的数据。其属性与方法见表 1-1。

表 1-1 ArrayList 的属性与方法

属性与方法	说　明
Count 属性	取得 ArrayList 中目前元素总数，例如：arraylist. Count( )
Length 属性	取得集合长度，例如：arraylist. Length( )
Add( )方法	将 obj 对象加到 ArrayList 的最后，例如：arraylist. Add(obj)
Insert( )方法	将 obj 对象插入到 ArrayList 指定的索引 count 的位置后面，其后元素依次往后移，例如：arraylist. Insert(count,obj)
Remove( )方法	由 ArrayList 删除第一个符合指定对象 obj 的元素，例如：arraylist. Remove(obj)
Clear( )方法	清除 ArrayList 中所有的元素，例如：arraylist. Clear( )
CopyTo( )方法	将 ArrayList 中全部或部分元素复制到另一个集合中
IndexOf( )方法	返回 ArrayList 中第一个符合指定对象 obj 的索引值。若找不到，则返回 -1。例如：arraylist. IndexOf (obj)
Sort( )方法	将 ArrayList 中所有元素以递增的方式排序，例如：arraylist. Sort( )
Reverse( )方法	将 ArrayList 中所有元素反转，例如：arraylist. Reverse( )
BinaruSerach( )方法	使用二分查找法由 ArrayList 找寻指定的对象 obj。若找不到，则返回一个负值。例如：arraylist. BinarySearch(obj)

【例 1-4】创建一个控制台应用程序，演示 ArrayList 的应用，如图 1-8 所示。

图 1-8 ArrayList 的应用示例

实现步骤如下：

1）在 VS 中创建一个名为 example1-4 的控制台应用程序。

2）在程序开头处输入 using System. Collections。

3）在 Main( )方法中添加如下代码：

```
ArrayList al = new ArrayList( );
int [ ]data  = new int[5] { 1, 3, 9, 8, 6 };
```

```
al. Add(10);     //添加一个元素
foreach (int number in data )
{
    al. Add(number);     //依次添加 data 数组中的 5 个元素
}
int[ ]number2 = new int[2] { 11, 12 };
al. AddRange(number2);     //添加 number 数组中的所有元素
al. Remove(3);     //移除值为 3 的元素
al. RemoveAt(3);     //移除第 3 个元素
Console. WriteLine("ArrayList 遍历:");
foreach (int i in al)
{
    Console. WriteLine(i);
}
Console. ReadLine( );
```

4）运行程序，观察结果。

2. 哈希表（HashTable）

HashTable 是一个 < 键（key）、值（value）> 对的集合，每一个元素都是这样一个对。这里，键（key）类似于数组中的下标，通过 key 可以唯一确定哈希表里的一个值（value）。

在哈希表中添加一个 key/value 键值对：HashtableObject. Add(key, value);

在哈希表中去除某个 key/value 键值对：HashtableObject. Remove(key);

从哈希表中移除所有元素：HashtableObject. Clear( );

判断哈希表是否包含特定键 key：HashtableObject. Contains(key);

遍历哈希表时，需要用到 DictionaryEntry 类。

【例 1-5】创建一个控制台程序，演示 HashTable 的应用，如图 1-9 所示。

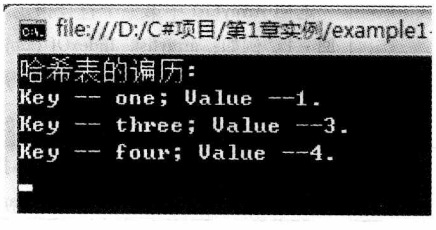

图 1-9　HashTable 的应用举例

实现步骤如下：

1）在 VS 中创建一个名为 example1-5 的控制台应用程序。

2）在程序开头处输入 using System. Collections;。

3）在 Main( )方法中添加如下代码：

```
Hashtable ht = new Hashtable( );     //创建一个 HashTable 实例
ht. Add("one", "1");                 //添加 key/value 键值对
ht. Add("two", "2");
```

```
ht. Add("three","3");
ht. Add("four", "4");
if (ht. Contains("two"))                    //判断哈希表是否包含特定键,其返回值为 true 或 false
ht. Remove("two");                          //移除一个 key/value 键值对
Console. WriteLine("哈希表的遍历:");
foreach (DictionaryEntry de in ht)
{
    Console. WriteLine("Key - - {0}; Value - - {1}.", de. Key, de. Value);
}
ht. Clear();                                //移除所有元素
Console. ReadLine();
```

4）运行程序，观察结果。

1.3.3 基本数据类型及转换

C#中最常用的数据类型，见表 1-2。

表 1-2 C#的常用数据类型

常用数据类型	特　征	取值范围	举　例
整型（int）	有符号 32 位整数	在 -2147483648～2147483647 之间	int val = 16
浮点型（float）	单精度浮点类型	在正负 1.5 × 10 - 45～3.4 × 1038 之间，精度为 7 位数	float val = 6.18F
字符串型（string）	表示一个 Unicode 字符序列，专门用于对字符串的操作	动态增长，与机器内存有关	string c = "hell"
布尔类型（boolean）	表示“真”和“假”这两个概念	true 和 false	bool x = true

在 C#语言中，数据类型之间可以相互转换。比如，从 int 类型转换到 long 类型。C#语言中，数据类型的转换可以分为两类：隐式转换和显式转换。

隐式转换就是系统默认的、不需要加以声明就可以进行的转换。对于数值类型，任何类型 Type1，其取值范围只要包含在类型 Type2 的取值范围内，就可以隐式转换为类型 Type2，比如从 int 类型可以隐式转换到 float、double 类型。隐式转换一般不会失败，转换过程中也不会导致信息丢失。

显式类型转换又叫做强制类型转换，与隐式转换正好相反，它明确地指定转换的类型。比如，把一个 long 类型显式转换为 int 类型。

```
long l = 5000;
int i = (int)l;
```

此外，还可以使用 Parse、ToString 方法以及 Convet 类进行类型转换。

1. Parse 方法

前面讲的显式和隐式类型转换都是在数值间进行的，那么，数值和字符串之间的转换就要用到 Parse( )方法了。

对于不同的数值类型都有自己的 Parse( )方法。

例如：字符串转换为整型（string→int）。

```
int. parse(string);
```

字符串转换为单精度浮点型（string→float）。

```
float. parse(string);
```

字符串转换为双精度浮点型（string→double）。

```
double. parse(string);
```

2. ToString 方法

将其他类型转换为字符串类型用 ToString 方法。例如，下面几条语句：

```
int i = 12345;
i. ToString( )          //结果是"12345"
double i = 12345. 6789;
i. ToString("f2")       //结果是 12345. 68
i. ToString("f")        //结果是 12345. 68
i. ToString("f4")       //结果是 12345. 6789
```

3. Convert 类

下面再来认识 Convert 类。它可以在各种基本类型之间互相转换。Convert 类为每种类型转换都提供了一个静态方法，见表 1-3。

表 1-3　Convert 类的方法

方法原型	说　明
Convert. ToBoolean(var)	var 转换为 bool
Convert. ToByte(var)	var 转换为 byte
Convert. ToChar(var)	var 转换为 char
Convert. ToDecimal(var)	var 转换为 decimal
Convert. ToDouble(var)	var 转换为 double
Convert. ToInt16(var)	var 转换为 short
Convert. ToInt32(var)	var 转换为 int
Convert. ToInt64(var)	var 转换为 long
Convert. ToSByte(var)	var 转换为 sbyte
Convert. ToSingle(var)	var 转换为 float
Convert. ToString(var)	var 转换为 string
Convert. ToUInt16(var)	var 转换为 ushort
Convert. ToUInt32(var)	var 转换为 uint
Convert. ToUInt64(var)	var 转换为 ulong

其中，var 可以是各种类型的变量。

【例 1-6】创建一个控制台程序，演示 Convert 类的应用，如图 1-10 所示。

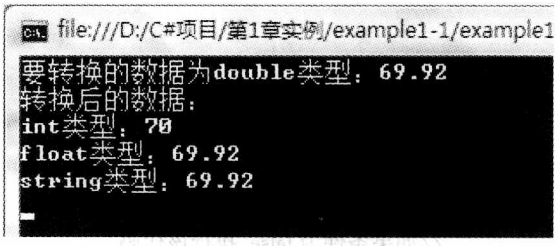

图 1-10　Convert 类的应用示例

实现步骤如下：

1）在 VS 中创建一个名为 example1-6 的控制台应用程序。

2）在 Main()方法中添加如下代码：

```
double dd = 69.92;                    //要转换的数据
int tt;                               //转换后的整型
float ff;                             //转换后的浮点型型
string ss;                            //转换后的字符串型
Console.WriteLine("要转换的数据为 double 类型:{0}", dd);  //开始转换
tt = Convert.ToInt32(dd);
ff = Convert.ToSingle(dd);
ss = Convert.ToString(dd);
//输出转换后的数据
Console.WriteLine("转换后的数据:");
Console.WriteLine("int 类型:{0}", tt);
Console.WriteLine("float 类型:{0}", ff);
Console.WriteLine("string 类型:{0}", ss);
Console.ReadLine();
```

3）运行程序，观察结果。

1.3.4　语句

任何程序代码的基本组成结构都是语句。所谓语句，是按照一定的语法规范来编写的。下面介绍 C#中几种常用的语句。

1. 条件语句

条件语句依据一个控制表达式的计算值从一系列可能被执行的语句中选择要执行的语句。条件语句主要有 if 语句和 switch 语句。

（1）if 语句

if 语句是最常用的选择语句。它根据所给条件是否满足，决定是否执行后面的操作。它有以下几种形式：

```
if (条件表达式)
{
    代码;                    //如果条件为 true,执行该代码,否则,直接执行后面的代码
}
```

```
if(条件表达式)
{
    代码;                    //如果条件为 true,执行该代码
}
else
{
    代码;                    //如果条件为 false,执行该代码
}
```

还可以使用 else if 进行多重 if 判断：

```
if(条件表达式 1)
{
    代码;                    //如果条件 1 为 true,执行该代码
}
else if(条件表达式 2)
    {
        代码;                //如果条件 2 为 true,执行该代码
    }
    else if(条件表达式 3)
    {
        代码;                //如果条件 3 为 true,执行该代码
    }
    else
    {
    代码;                    //如果都不成立,执行该代码
    }
```

【例 1-7】 对一个浮点数 f 进行四舍五入，结果保存到一个整数 i 中。

实现步骤如下：

1）在 VS 中创建一个名为 example1-7 的控制台应用程序。

2）在 Main( )方法中添加如下代码：

```
int i;
double f = 2.9;
if ((f - (int)f) > 0.5)
{
    i = (int )f +1;
}
else
{
    i = (int )f;
}
Console. WriteLine(i);
Console. ReadLine( );
```

3）这段代码输出为3。

（2）switch 语句

switch 语句依据表达式的值选择执行相关的语句。switch 语句可以生成比用多个 if 语句更简洁的代码，因为控制只需计算一次。

【例 1-8】输入一个给定的 1 ～12 之间的任意整数，返回这个数字所对应月份的英文单词，如图 1-11 所示。

实现步骤如下：

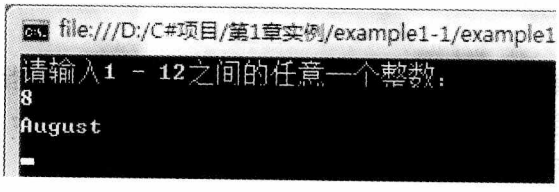

图 1-11　switch 语句应用示例

1）在 VS 中创建一个名为 example1-8 的控制台应用程序。

2）在 Main( )方法中添加如下代码：

```
Console. WriteLine("请输入 1 ～12 之间的任意一个整数:");
string s = Console. ReadLine( );
int month = int. Parse(s);                    //把输入的字符串转换成一个整型数
switch (month)
{
    case 1: Console. WriteLine("January"); break;
    case 2: Console. WriteLine("February"); break;
    case 3: Console. WriteLine("March"); break;
    case 4: Console. WriteLine("April"); break;
    case 5: Console. WriteLine("May"); break;
    case 6: Console. WriteLine("June"); break;
    case 7: Console. WriteLine("July"); break;
    case 8: Console. WriteLine("August"); break;
    case 9: Console. WriteLine("September"); break;
    case 10: Console. WriteLine("October"); break;
    case 11: Console. WriteLine("November"); break;
    case 12: Console. WriteLine("December"); break;
    default: Console. WriteLine("输入错误,转换失败。"); break;
}
Console. ReadLine( );
```

3）执行程序。当输入为 8 时，输出结果是 August。

2. 循环语句

循环语句重复执行某一组语句。C#中主要有 for 语句、while 语句、do 语句和 foreach 语句。

（1）for 语句

for 语句是 C#中使用频率最高的循环语句。在已知循环次数的情况下，使用 for 语句是比较方便的。for 语句首先初始化循环变量，当条件表达式的值为 true 时，执行语句。例如如下代码：

```
for(int i = 1;i <= 10;i ++)
    {
        Console. writeline(i);
    }
```

这段代码输出 1 ～10。

（2）while 语句

当条件表达式的值为 true 真时，while 语句将一直执行，直到指定的条件表达式的值为 false 时为止，可以使用 break、goto 等语句终止循环。使用 continue 语句可以执行下一个循环而不退出循环。例如如下代码：

```
int i = 5;
while (i <= 100)
{
    i += 10;
}
Console. writline(i);
```

i 值初始化为 5，每次增加 10，最终 i 的值为 105 时退出循环。

（3）do…while 语句

与 while 语句不同，由于 do…while 语句是先执行后判断条件，所以它的循环体至少执行一次。例如如下代码：

```
int i = 0;
do
  {
    Console. WriteLine(i);
    i += 10;
  }
    while (i ! = 100);
```

这段代码输出 0 ～100。

（4）foreach 语句

foreach 语句是在 C#中新引入的，each 是每一个的意思，那么，foreach 就是循环每一个。它表示对一个数组或集合中的每个元素重复执行一个代码块。

【例 1-9】 创建一个控制台程序，计算一维整型数组中奇数的个数和偶数的个数，如图 1-12 所示。

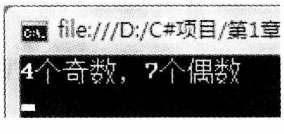

图 1-12　foreach 语句应用示例

实现步骤如下：

1）在 VS 中创建一个名为 example1-9 的控制台应用程序。

2）在 Main( )方法中添加如下代码：

```
int odd = 0, even = 0;
int[ ] array = new int[ ] { 0, 4, 7, 8, 9, 10, 14, 17, 19, 24, 56 };
foreach (int i in array)
{
    if (i % 2 = = 0)
        even ++;
    else
      odd ++;
}
Console. WriteLine("{0}个奇数,{1}个偶数", odd, even);
Console. ReadLine( );
```

3）运行程序，观察结果。

3. 跳转语句

跳转语句是中断程序的顺序执行，无条件地转入到另一个地方继续执行。C#语言中提供了 break 和 continue 跳转语句。

（1）break 语句

break 语句终止 switch、while、do…while、for、foreach 等语句在 break 出现的那一层次的运行，紧接着该层外的下一条语句继续执行。

【例 1-10】依次输出 1 ～10，当遇到 5 就停止输出，如图 1-13所示。

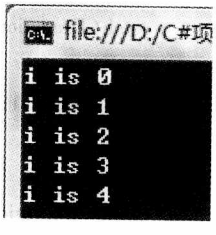

图 1-13　break 语句应用示例

实现步骤如下：

1）在 VS 中创建一个名为 example1-10 的控制台应用程序。

2）在 Main( )方法中添加如下代码：

```
for (int i = 0;i < 10;i ++)
{
    if (i! = 5)
        Console. WriteLine("i is " + i);
    else
        break;
}
Console. ReadLine( );
```

3）运行程序，观察结果。

（2）continue 语句

continue 语句终止 while、do…while、for、foreach 等语句在 continue 出现的那一层次的当前循环，执行新一次循环。

【例 1-11】依次输出 10 以内除 5 以外的数字，如图 1-14 所示。

图 1-14　continue 语句应用示例

实现步骤如下：

1）在 VS 中创建一个名为 example1-11 的控制台应用程序。

2）在 Main( )方法中添加如下代码：

```
for (int i =0;i <10;i ++)
{
    if (i ! =5)
        Console. WriteLine("i is " +i);
    else
        Continue;
}
Console. ReadLine( );
```

3）运行程序，观察结果。

4. 异常处理

在程序运行时可能会发生各种不可预期的情况，比如，用户输入错误、内存不够、磁盘出错、网络资源不可用、数据库无法使用等，所有这些错误被称为异常，不能因为这些异常使程序运行产生问题。在 C#语言中，异常处理是使用 try…catch…finally 语句来完成的。它提供一种机制，用于捕捉在 try 语句块的执行期间发生的各种异常。try…catch…finally 语句的语法格式如下：

```
try
{
    //受监视的程序代码
}
catch(Exception1 e)
{
    //捕捉并处理异常类型 1
}
  …
catch (ExceptionN e)
{
    //捕捉并处理异常类型 2
}
finally
{
    //最后执行的程序代码
}
```

try 语句：我们一般把可能会出现异常的语句放在 try 子句中，“try 语句块”中的语句会逐条运行，直到完成或者产生异常。

catch 语句：若发生异常，则会按顺序检查每个 catch 语句，以确定它是否处理异常。

finally 语句块：是执行过程离开 try 语句的任何部分时，总会执行 finally 语句块的。

Exception e：定义了一个 Exception 异常类的对象 e。在 C#中，所有的异常都继承于 Exception 异常类。System. Exception 命名空间中常用的几种异常类见表 1-4。

表 1-4　常用的异常类

异　常　类	说　　明
ArgumentOutOfRangeException	当参数值超出调用方法规定的范围时产生错误
DivideByZeroException	除数为 0 时产生错误
Exception	程序运行时产生错误
IndexOutOfangeException	索引值超出数组运行范围
InvalidCastException	类型转换时产生错误，如将字母转为数值
OverFlowException	溢出时产生错误

若“try 语句块”运行时产生异常，异常处理的执行过程如下：

try 语句块在发生异常的地方中断程序的运行。如果有 catch 语句，就检查该语句是否匹配所发生的异常类型，如果匹配，则运行 catch 子句，异常将在 catch 语句中被处理。允许使用多个 catch 语句，发生异常时，会按顺序检查每个 catch 语句。如果有 finally 语句，就继续执行 finally 语句中的代码。如果没有与发生异常相匹配的 catch 语句，则直接执行 finally 语句中的代码（如果没有 catch 语句，则必须有 finally 语句）。

【例 1-12】异常处理的简单应用：当除数为 0 时产生异常，代码如下。

1）在 VS 中创建一个名为 example1-12 的控制台应用程序。

2）在 Main( )方法中添加如下代码：

```
int x = 5, y = 0;
try
{
    x / = y;
}
catch (DivideByZeroException e)
{
    Console. WriteLine(e. Message);
}
finally
{
    Console. WriteLine("\n 无论异常是否发生,finally 子句每次都被执行");
}
Console. ReadLine( );
```

3）运行程序，观察结果，如图 1-15 所示。

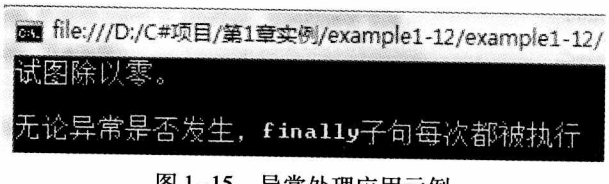

图 1-15　异常处理应用示例

1.3.5　命名空间

C#程序是使用命名空间来组织的。命名空间（NameSpace）是 .NET 类库的逻辑分区。

每个命名空间都包含一组按照功能划分的相关的类。这样，一个大型的 .NET 库就变得易于理解和便于使用，将逻辑上相关的类组织到一起，以便于查找和引用。

.NET 框架类库的根命名空间 system 包含了所有基类型对象，所有其他类型都从基类型继承而来。表 1-5 列出了 C#中常用的命名空间。

表 1-5　C#中常用的命名空间

命 名 空 间	说　明
System. Drawing	用于处理图形和绘图，包括打印
System. Data	用于处理数据存取和管理，在定义 ADO. NET 技术中扮演重要角色
System. IO	用于管理对文件和流的同步和异步访问
System. Windows	用于处理基于窗体的窗口的创建
System. Reflection	包含从程序集读取元数据的类
System. Threading	包含用于多线程编程的类
System. Collections	包含定义各种对象集的接口和类

在程序设计中，命名空间主要是为了解决 C#中变量、函数的命名冲突而服务的。可以将同一名称的变量或函数定义在不同的命名空间中，比如，王家有计算机，张家也有计算机，我们识别谁家的计算机的时候，只需要用所属的家庭作为前缀就可以了。

定义命名空间时使用关键字 namespace，下面定义一个命名空间：

```
namespace zhangjia                    //声明张家的计算机
{
    class computer
    {
        public string memory;
        public string cpu;
    }
}
```

上面这段代码定义了一个命名空间“zhangjia”，其中定义了一个计算机类，类中添加了两个字段信息。用 zhangjia. computer 就可以表示张家的计算机了。

命名空间是可以嵌套的，嵌套的命名空间通过层次结构来引用，并用“.”来区分结构的层次。例如：

```
namespace Level1
{
    class computer
        {
            // computer 类的声明语句
            …
        }
namespace Level2
{
    class computer
    {
```

```
        // computer 类的声明语句
        ...
    }
}
```

上面的代码中，用 Level1. computer 和 Level2. computer 来区分 computer 类，这样就不会有命名冲突了。如果在 Level1 外使用 Level2 里的 computer 类就要用 Level1. Level2. computer 表示了。

1.4 C#面向对象编程

C#是一种面向对象的编程语言，对象是 C#的核心。在这种语言中，执行所有的操作几乎都与对象相关联。下面给出 C#面向对象编程的基本概念。

1.4.1 类及类的实例

类与对象是面向对象编程的基本概念。类是一组具有共同特征的内容的高度抽象表示形式，在类中封装了对象包含的信息（如字段、属性）和对象可以执行的操作（即方法），而对象是类的实例。

类（class）是一个类型的声明，比如，每一辆汽车是一个对象的话，所有的汽车可以作为一个模板，我们就定义汽车这个类。C#使用 class 关键字来定义类的，语法格式如下：

```
[访问级别] class [类名]:[父类]
{
    [类的成员]
}
```

类在内存中并不存在，而对象是类的一个实例，对象在内存中占有一定的存储空间，可以使用类中提供的字段和方法。有了类的定义，用户可以使用 new 关键字为类创建一个实例。比如下面代码：

```
Class Student
{
    public int age;
    public int ComputeAge( )
    {
        Student stu = new Student ( );
    }
}
```

先定义了一个类 Student，然后在类中定义一个整数 age 和方法 ComputeAge，在方法 ComputeAge 中创建了一个类 Student 的实例 stu。

1.4.2 类的构造函数

每当创建类的对象时，会自动调用类的构造函数。因此，可把初始化工作放到构造函数中完成。构造函数和类名相同，没有返回值，访问级别为 public。