第4章 类型转换和数组

4.1 项目导引——游戏平台积分排行榜算法的实现

通过上一章的学习，我们开发出一个简易的面向过程的控制台版21点游戏，了解了C#中的流程控制及游戏的简单逻辑，如前所述，21点游戏是游戏平台的一部分，而在平台上注册的玩家，都要记录其积分，且应该对积分进行排序，形成积分排行榜。在本章项目主要是利用数组及结构实现积分排行榜的算法。

4.2 项目构思

项目描述：编写积分排行榜程序，实现积分排行。

项目要求：

(1) 使用结构类型的数组保存玩家名称、玩家积分。

(2) 程序启动后，通过控制台输入玩家的姓名及积分。

(3) 在选择排行开始之后，将排行信息输出到控制台。

4.3 项目分析

1. 开发思路

要想实现两个数的加法运算，按照以下几个步骤进行就可以完成任务。

(1) 定义结构类型的数组。

(2) 从控制台输入玩家信息，并保存在数组中。

(3) 运行程序，计算机将运行结果显示在屏幕上。

实际上，以上各步骤的实现都是在上章所介绍的流程控制基础上完成的，只是用C#进行编码之前，我们需要考虑一个问题：玩家信息应至少包含玩家的姓名和积分，而姓名和积分通常是不同的数据类型，在同时大批量的输入不同的数据类型数据时，计算机应如何存储？

在编码之前，先来学习涉及到的新知识。

2. 新知识点

（1）类型转换。

在高级语句中，数据类型是很重要的一个概念，只有具有相同数据类型的对象才能够相互操作。很多时候，为了执行不同类型数据的运算，需要把数据从一种类型转换为另一种类型，这就是所谓的类型转换。

（2）数组的定义。

数组是一种包含若干变量的数据结构，这些变量都具有相同的数据类型并且排列有序，因此可以用一个统一的数组名和下标唯一地确定数组中的元素。C＃中的数组主要有三种形式：一维数组、多维数组和不规则数组。

```
数据类型　数组名[ 整数值 ];
```

（3）数组元素的遍历。

通过“数组名［下标］”的形式访问一维数组中的成员。这里，下标的范围是有限制的，必须在 0 到数组元素个数－1 之间，超出这个范围的限制，会导致 IndexOutOfRangeException 异常。

（4）数组元素的排序。

按照一种排序方法进行排序，使用某种循环结构实现。

4.4　项目实施

（1）玩家信息的定义如下：

```
struct Player
{
    public string PlayerName;
    public int Score;
}
```

（2）玩家信息输入代码如下：

```
static void Main(string[] args)
{
    Console.Write("请输入玩家人数:");
    int PlayerNum=Convert.ToInt32(Console.ReadLine());
    Player[] playerinfo=new Player[PlayerNum];
    for (int i=0; i < PlayerNum; i++)
    {
        Console.Write("输入第{0}玩家信息,玩家姓名及积分,用空格分隔",i+1);
        string pf=Console.ReadLine();
        Player player;
        string[] aplayer=pf.Split(' ');
        player.PlayerName=aplayer[0];
```

```
        player.Score=Convert.ToInt32(aplayer[1]);
        playerinfo[i]=player;
}
```

(3) 玩家积分排行榜的实现如下，其排序算法为冒泡排序算法。

```
for (int i=0; i<PlayerNum; i++)
    {
        for (int j=PlayerNum - 1; j > i; j--)
        {
            if (playerinfo[j].Score > playerinfo[j - 1].Score)
            {
                // 第 j 与第 i 个玩家信息交换,把积分高的玩家向上推
                player=playerinfo[j];
                playerinfo[j]=playerinfo[j - 1];
                playerinfo[j-1]=player;

            }
        }
    }
    for (int i=0; i < PlayerNum;i++)
    {
        Console.WriteLine("玩家姓名:{0}    玩家积分:
        {1}",playerinfo[i].PlayerName,playerinfo[i].Score);
    }
```

(4) 使用 Ctrl+F5 运行程序，可以得到如图 4-1 所示的结果。

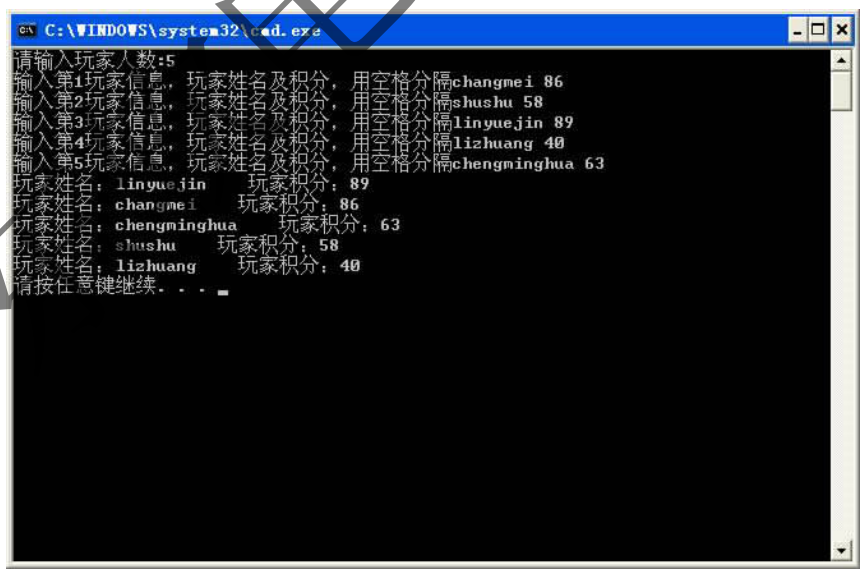

图 4-1 积分排行榜程序运行结果

4.5　知识点详解

4.5.1　类型转换

类型转换方法分为隐式类型转换和显式类型转换，也可以采用 Convert 类的方法实现类型转换。

1. 隐式类型转换

隐式类型转换：从类型 A 到类型 B 的转换可以在所有情况下进行，执行转换的规则十分简单，编辑器自动将操作数转换为相同类型。转换的时候不需要加任何声明就可以实现。通常在整数类型、实数类型和字符类型之间的转换就属于隐式类型转换。例如：

```
sbyte a=100;
int b=a;
```

隐式类型转换如表 4-1 所示。

表 4-1　　隐式类型转换

原始类型	可转换到的类型	可能有信息丢失
sbyte	short，int，long，float，double，decimal	
byte	short，ushort，int，uint，long，ulong，float，double，decimal	
short	int，long，float，double，decimal	
ushort	int，uint，long，ulong，float，double，decimal	
int	long，float，double，decimal	float
uint	long，ulong，float，double，decimal	float
long	float，double，decimal	float，double
ulong	float，double，decimal	float，double
char	ushort，int，uint，long，ulong，float，double，decimal	
float	double	

从 int 到 long，从 long 到 float、double 等几种类型转换时可能导致精度的下降，但不导致信息的丢失。任何的原始类型，如果值的范围完全包含在其他类型值的范围内，那么就能进行隐式转换。char 类型可以转换为其他的整数或实数类型，但不存在其他类型转换为 char 类型。

【实验 4-1】隐式类型转换示例。

内容：使用隐式转换，将类型“char”转换为“int”。

实现：

(1) 启动 Visual Studio 2010，创建控制台应用程序 Demo4-1，并保存到适当的位置。

(2) 在 Main () 函数中填写如下代码：

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

```
{
    char a='m';
    int b=a;
    Console.WriteLine("a equals:{0}",a);
    Console.WriteLine("b equals:{0}", b);
}
```

(3) 使用 Ctrl+F5 启动程序，得到如图 4-2 所示的结果。

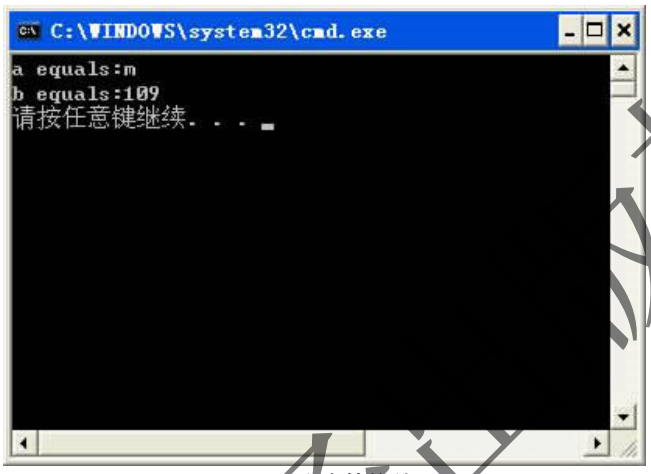

图 4-2 隐式转换结果

分析：

(1) b 中的值是字符 m 的 ASCII 码。

(2) 从示例代码中可以看出，字符类型到整型可以隐式地转换。

(3) 但整型无法隐式地转换成字符类型，下面的代码将导致编译错误：无法将类型“int”隐式转换为“char”。

```
int b=7;
char a=b;
Console.WriteLine("a equals:{0}",a);
Console.WriteLine("b equals:{0}",b);
```

编译器将报错：无法将类型“int”隐式转换为“char”。

2. 显式类型转换

在某些情况下，编辑器不能隐式转换数据类型，必须进行显式转换。显式转换规则复杂，需要用户正确指定要转换的类型，又称强制类型转换。例如：

```
int b=100;
sbyte a=b;
```

上述两行代码视图将 int 类型的变量转换为 sbyte 类型的变量，放在 main 函数中运行时，将提示错误：无法将类型“int”隐式转换为“sbyte”，如图 4-3 所示。

图 4-3 提示无法实现隐式转换

但由于某种原因，开发人员可能必须要执行这样的操作，这时就必须使用显式转换数据类型强迫编辑器进行转换。其转换的一般语法如下所示：

(要转换的目的数据类型)源数据；

在显式转换时，开发人员需要了解所要转换的数据类型，保证在转换之后不丢失数据。显式数据类型转换如表 4-2 所示。

表 4-2　　显式类型转换

原始类型	可转换到的类型
sbyte	byte，ushort，uint，ulong，char
byte	sbyte，char
short	sbyte，byte，ushort，uint，ulong，char
ushort	sbyte，byte，short，char
int	sbyte，byte，short，ushort，uint，ulong，char
uint	sbyte，byte，short，ushort，int，char
long	sbyte，byte，short，ushort，int，uint，ulong，char
ulong	sbyte，byte，short，ushort，int，uint，long，char
char	sbyte，byte，short
float	sbyte，byte，short，ushort，int，uint，long，ulong，char，decimal
double	sbyte，byte，short，ushort，int，uint，long，ulong，char，float，decimal
decimal	sbyte，byte，short，ushort，int，uint，long，ulong，char，float，double

【实验 4-2】显式类型转换示例。

内容：使用显式转换，将类型“char”转换为“short”。

实现：

(1) 启动 Visual Studio 2010，创建控制台应用程序 Demo4-2，并保存到适当的位置。

(2) 在 Main() 函数中填写如下代码：

```
static void Main(string[] args)
{
    short s;//定义一个 short 类型变量 s
    char c='a';//定义一个 char 类型变量 c
    s=(short)c;//使用显式转换
    Console.WriteLine("字符:{0}",c);
    Console.WriteLine("对应的 Ascill 码:{0}",s);
}
```

(3) 使用 Ctrl+F5 启动程序，得到如图 4-4 所示的结果。

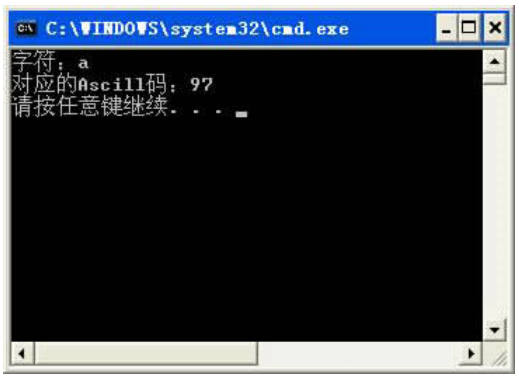

图 4-4 显式转换结果

分析：

字符存储于内存实际上保存的是字符的 unicode 码，对于英文字母而言，ASCII 码与 unicode 码的值是一致的。因此，字符与整数类型之间可以实现相互的转换。字符可以隐式地转换成整型，但其他整数类型与字符之间的相互转换，以及整型到字符类型的转换都需要显式地执行。

3. 使用 Convert 转换

System. Convert 类中有一套静态方法实现类型的转换，即使要转换的类型之间没有什么联系也可以很方便地实现类型的转换。Convert 类包含的类型转换方法如表 4-3 所示。

表 4-3 Convert 包含的类型转换方法

方法	实现的转换类型	方法	实现的转换类型
Convert. ToBoolean（）	bool	Convert. ToInt32（）	int
Convert. ToByte（）	byte	Convert. ToInt64（）	long
Convert. ToChar（）	char	Convert. ToSByte（）	sbyte
Convert. ToString（）	string	Convert. ToSingle（）	float
Convert. ToDecimal（）	decimal	Convert. ToUnit16（）	ushort
Convert. ToDouble（）	double	Convert. ToUnit32（）	unit
Convert. ToInt16（）	short	Convert. ToUnit64（）	ulong

无法产生有意义结果的转换将引发异常，运行结果不执行任何转换，异常通常为 OverflowException 或 FormateException，可以用异常处理块 try/catch 捕获和处理异常。

例如：

```
String str="32767";
System.Convert.ToInt16(str);
```

可以进行正常的类型转换。但是，

```
String str="32768";
System.Convert.ToInt16(str);
```

或：

```
String str="";
System.Convert.ToInt16(str);
```

要转换的字符串的值超过 short 的范围或不是数值都会导致异常。下面实验中将体会。

【实验 4-3】 System. Convert 实现类型转换示例。

内容： 使用 Convert 类实现将从控制台的输入的任意类型转换成 int 类型。

实现：

(1) 启动 Visual Studio 2010，创建控制台应用程序 Demo4-3，并保存到适当的位置。

(2) 在 Main () 函数中添加如下示例代码：

```
static void Main(string[] args)
{
    int a;
    string s;
    Console.WriteLine("Enter a string:");
    s=Console.ReadLine();//从控制台输入,假设输入字符串"123"
    a=Convert.ToInt32(s);//使用 Convert 类将从控制台输入的字符串转换为 int 类型
    Console.WriteLine("a equals:{0}",a);
}
```

(3) 使用 Ctrl+F5 启动程序，得到如图 4-5 所示的结果。

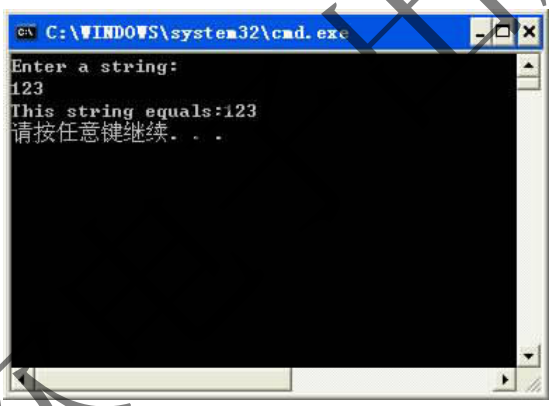

图 4-5　使用 Convert 类转换

分析：

(1) 在上面的示例代码中，通过 Console. ReadLine () 从控制台读取用户的输入，并保存到字符串 s 中。

(2) 使用 Convert. ToInt32 () 方法可以将该字符串转换成对应的整型数字。

(3) 当用户的输入不是整型数字时，会引发 FormateException 异常；当用户输入整数过大，超出整型表示的范围时，引发 OverFlowException。

4.5.2　数组

1. 一维数组

一般而言，数组都必须先声明后使用，在 C/C++这类语言中，数组在声明时，就要明确数组的元素个数，由编译器分配存储空间，但在C＃中数组是一个引用型类型，声明数组时，只是预留一个存储位置以引用将来的数组实例，实际的数组对象是通过 new 运算符在运行时动态产生的。因此，在数组声明时，不需要给出数组的元素个数。

(1) 一维数组的声明。

C# 中声明一维数组的语法格式如下：

```
元素类型名[ ] 数组名;
```

例如，下面代码声明了一个各元素都是整型的数组 intArray：

```
int[ ] intArray;
```

(2) 初始化一维数组中的元素值。

①数组名＝new 数组元素的类型［常数/常量/变量］;

在声明了对应的数组之后，可以使用上面的方法初始化数组元素，这样初始化的数组中各元素的值都相等，均为该类型元素的默认值。例如，数值类型的默认值为 0，字符类型的默认值为空格''，引用类型的默认值为 null。

②可以在声明数组的同时，直接为数组元素指定初始值，赋值的方法是用一个逗号分隔的元素值列表，列表用大括号括起来。例如：

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

上面的代码声明了一个含有 5 个整型数据的一维数组，并在声明的同时为各数组元素赋值依次为 1，2，3，4 和 5。

③可以用①和②结合的方法初始化数组。例如：

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

需要注意的是，new 数据类型后面的 [] 中的值是可以省略的，但如果没有省略，该值必须与后面 {} 中元素的个数相等；此外，如果没有省略，该处的值只能用常数或常量表示，不能在该处使用变量。

(3) 访问一维数组中的成员。

通过“数组名［下标］”的形式访问一维数组中的成员。这里，下标的范围是有限制的，必须在 0 到数组元素个数－1 之间，超出这个范围的限制，会导致 IndexOutOfRangeException 异常。

使用 for 和 foreach 循环遍历一维数组。

①使用 for 循环遍历一维数组：

```
for(int i=0;i<数组长度;i++)
{
    对 数组名[i] 的访问代码;
}
```

这里，数组的长度即数组中元素的个数，可以通过“数组名 . Length”属性获取。

②使用 foreach 循环遍历一维数组：

```
foreach(数组元素的数据类型 变量名 in 数组名)
{
对 变量名 的读访问代码;
}
```

使用 foreach 的过程中，不允许对所访问的数组做任何的修改。

【实验 4-4】 定义一个数组，存放一组数据，找出该数组中最大数和最小数。

内容： 通过下标访问数组，将数组中的最大数和最小数分别在控制台输出。

设计： 根据实验内容要求，设计程序流程如图 4-6 所示。

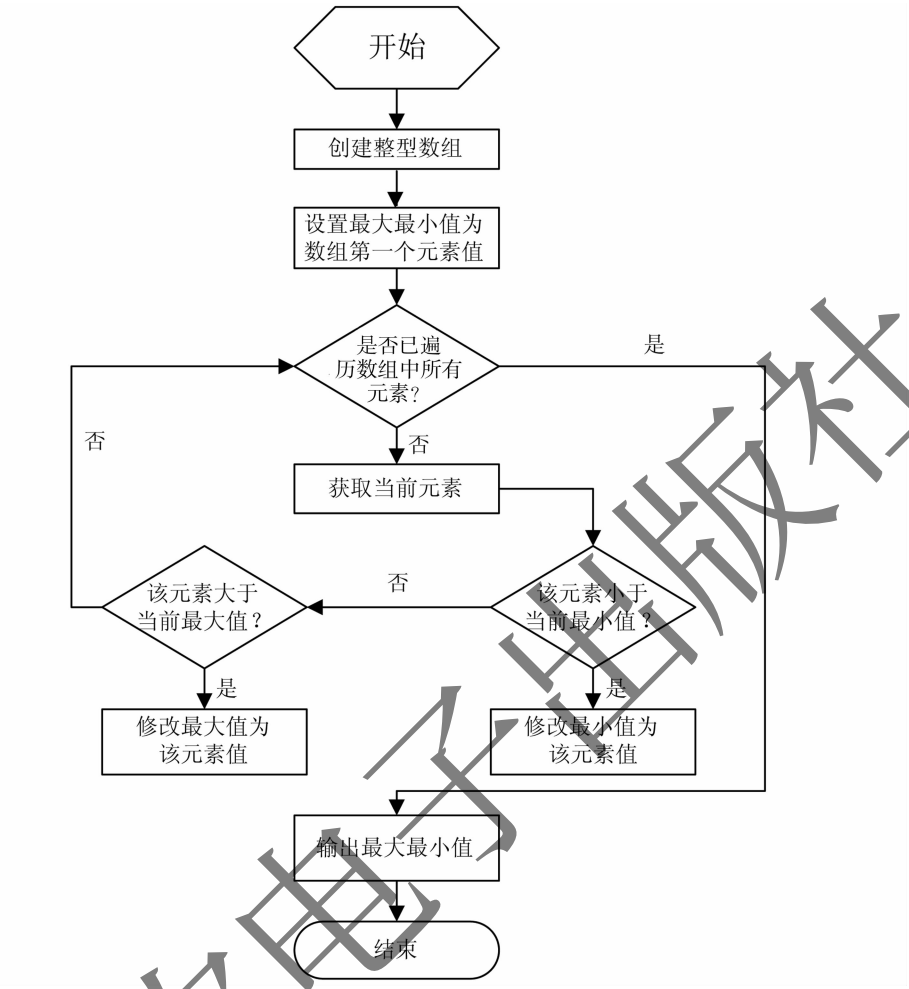

图 4-6　【实验 4-4】流程图

实现：

(1) 启动 Visual Studio 2010，创建控制台应用程序 Demo4-4，并保存到适当的位置。

(2) 根据流程图，在 Main () 函数中添加如下代码：

```
static void Main(string[] args)
{
    int max, min;
    int[] queue=new int[10] { 89,78,65,52,90,92,73,85,91,95};
    max=min=queue[0];
    foreach (int i  in  queue) {//使用 foreach 语句遍历数组元素
        if (i> max) max=i;
        else if (i < min) min=i;
    }
    Console.WriteLine("最大数是{0},最小数是{1}",max,min);
}
```

(3) 使用 Ctrl+F5 启动程序，得到如图 4-7 所示的结果。

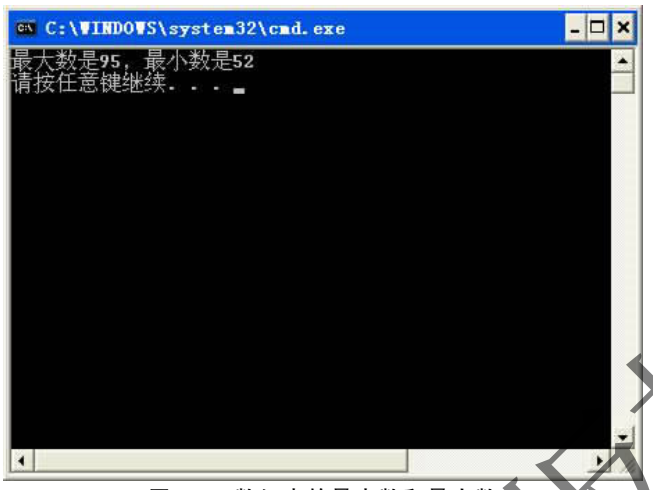

图 4-7 数组中的最大数和最小数

分析：

（1）根据内容要求，需要对数组中的每个元素都执行访问，因此考虑使用 for 或 foreach 循环。

（2）根据流程图的设计，在遍历数组元素的过程中，只是读取元素值，并不修改元素值，因此使用 foreach 循环实现。

2. 二维数组

数组可以具有多个维度，多维数组中比较常用的是二维数组。

（1）二维数组的声明。

```
数组元素类型[,]  数组名称;
```

当有多个逗号时，即为多维数组。

例如，下面代码声明了一个二维 double 类型的数组。

```
double[,] lengths;
```

（2）初始化二维数组。

初始化二维数组的方法与一维数组类似，主要包括如下几种方法：

①在声明的同时赋予默认值，语法如下：

```
数组元素类型[,]  数组名称=new 数组元素类型[一维长度,二维长度];
```

例如，下面代码在声明的同时初始化了一个 double3 行 4 列的二维数组 doubleArray，每个元素的初始值都是 double 类型的默认值 0：

```
double[,] doubleArray=new double[3,4];
```

②在声明的同时赋予各元素指定值，语法如下：

```
数组元素类型[,]  数组名称=new 数组元素类型{各元素的值};
```

则数组元素中的值即对应直接指定的值。例如：

```
int[,] intArray1={ { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

需要注意的是，这种直接赋值的方法只能在声明的同时进行，否则在赋值时需要先指定各维度的长度。

③如果选择声明一个数组变量但不将其初始化，必须使用 new 运算符将一个数组分配给此变量。例如：

```
int[,] intArray2;
intArray2=new int[4,2]{ { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

这里，new int［4，2］中的 4 和 2 分别是二维数组 2 个维度的长度，它们是可省的，也可以在这个位置上用常量或变量替换常数。需要注意的是如果这里使用常量或变量，其值必须和后面的元素个数相对应。

(3) 访问二维数组中的元素。

访问二维数组中的指定元素仍然是通过数组名和下标来访问的，二维数组中的第 i 个元素的访问方法为，数组名［(i－1) /第二个维度的长度，(i－1)％第二个维度的长度］。例如：

```
int[,] intArray3=new int[3,4];
intArray3[1,2]=2;
```

上面的代码中声明并初始化了一个 3 行 4 列的整型二维数组，初始化时每个元素的值都是 0；在第二行代码中，将数组中第 7 个元素赋值为 2。

需要注意的是，在访问二维数组的元素时，如果还没有初始化该数组，或者越界访问，都可能引发异常。

(4) 遍历二维数组。

①使用 for 循环控制语句遍历二维数组。

使用 for 循环控制语句遍历二维数组时，需要通过循环的嵌套来实现。外层循环遍历数组的行数，内层循环遍历数组的列数。一般的方法为：

```
for(int i=0;i<第一个维度的长度;i++)
{
    for(int j=0;j<第二个维度的长度;j++)
    {
        对 数组名[i,j] 的访问代码;
    }
}
```

例如，下面代码可以为 intArray3 中的每个元素赋值，将所有元素的值都设置为 10。

```
for(int i=0;i<3;i++)
{
    for(int j=0;j<4;j++)
    {
        intArray3[i,j]=10;
    }
}
```

②使用 foreach 循环控制语句遍历二维数组。

使用 foreach 循环控制语句遍历二维数组的方法和遍历一维数组的方法是一样的。

```
foreach(元素的数据类型 变量名 in 数组名)
{
    对 变量名 的读操作代码;
}
```

使用 for 循环遍历二维数组时需要使用双层循环，但使用 foreach 循环时，只需要单层

循环。

【实验 4-5】求两个矩阵的乘积。

内容：定义一个 3 行 4 列的整型的二维数组 a 表示矩阵 A，定义一个 4 行 3 列的整型的二维数组 b 表示矩阵 B，定义一个 3 行 3 列的整型的二维数组 c 表示矩阵 A 和矩阵 B 相乘后得到的新矩阵 C。数组定义的同时进行初始化。求出矩阵 C 并将其在控制台以 3 行 3 列的矩阵形式输出。

实现：

(1) 启动 Visual Studio 2010，创建控制台应用程序 Demo4-5，并保存到适当的位置。

(2) 在 Main () 函数中添加如下示例代码：

```
static void Main(string[] args)
{
    int i, j, k;
    //定义一个 3 行 4 列的整型的二维数组 a 表示矩阵 A
    int[,] a=new int[3, 4]{ { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 } };
    //定义一个 4 行 3 列的整型的二维数组 b 表示矩阵 B
    int[,] b=new int[4, 3]{ { 12, 11, 10 }, { 9, 8, 7 }, { 6, 5, 4 }, { 3, 2, 1 } };
    //定义一个 3 行 3 列的整型的二维数组 c 表示矩阵 A 和矩阵 B 相乘后得到的新矩阵 C
    int[,] c=new int[3, 3];
    //用循环实现两个矩阵的相乘
    for (i=0; i < 3; i++)
    {
        for (j=0; j < 3; j++)
    {
        for (k=0; k < 4; ++k)
        {/*c[i,j]的值为数组 a 的第 i 行的每个元素分别与数组 b 的第 j 列的每个元素相乘后再
相加的和*/
            c[i, j] +=a[i, k] * b[k, j];
        }
    }
}
    for (i=0; i < 3; ++i)
        or (j=0; j < 3; ++j)
        {
            Console.Write("{0,4:d}", c[i, j]);
        }
            Console.WriteLine();
        }
}
```

(3) 使用 Ctrl+F5 启动程序，得到如图 4-8 所示的结果。

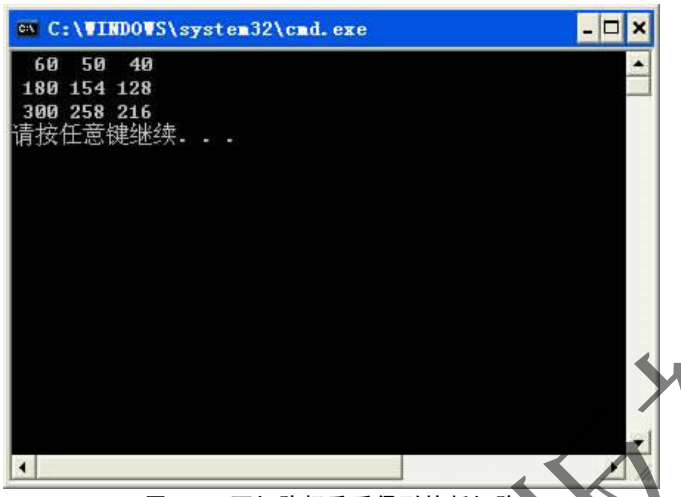

图 4-8　两矩阵相乘后得到的新矩阵

分析：

（1）本实验中使用了三层循环的嵌套。

（2）最外层的循环次数等于新生成数组的行数，该值应该等于第一个数组的行数和第二个数组的列数。针对当前实验，循环次数等于 3，也可以使用 a.GetLength（0）或 b.GetLength（1）来代替。

（3）内部嵌套了两个双层循环，第一个循环用于数组元素的计算；第二个循环用于数组元素的输出。

3. 交错数组

交错数组是元素为数组的数组。交错数组元素的维度和大小可以不同。交错数组有时称为“数组的数组”，数组中的每个元素又是一个数组。

（1）交错数组的声明。

声明数组的数组语法为：

```
数据类型[][] 数组名;
```

例如，“int [] [] jaggedArray;”声明了一个整型的交错数组 jaggedArray，该数组包含了未知个数的元素，每个元素的数据类型都是一维的整型数组。

（2）初始化交错数组。

交错数组也必须初始化后元素才可以访问。初始化的方法主要包括如下几种：

初始化交错数组的一般语法如下：

```
数组元素类型[][]  数组名称＝new 数组元素类型[元素个数][]
{
    new 数组元素类型[]{元素值 1,元素值 2,…元素值 m1},
    new 数组元素类型[]{元素值 1,元素值 2,…元素值 m2},
    …
    new 数组元素类型[]{元素值 1,元素值 2,…元素值 mn}
};
```

其中，元素个数是可以省略的，等于后面 {} 中元素的个数，new 后面第二个 [] 中不能填写任何数字，因为每个元素又是一个新的数组，这些数组的长度可能是不一致的。

例如：

```
int[][] jaggedArray2＝new int[][]
{
    new int[] {1,3,5,7,9},
    new int[] {0,2,4,6},
    new int[] {11,22}
}
```

jaggedArray2 与 jaggedArray 的数据类型相同，但其值在声明的同时就得到了初始化；jaggedArray2 中包含 3 个元素，每个元素都是一个一维整型数组；第一个元素中包含了 5 个整型元素，第二个元素中包含了 4 个整型元素，第三个元素中包含 2 个整型元素。

(3) 在声明的同时初始化交错数组。

如果初始化操作是在声明的同时发生的，则上面的 new 数据类型 [] [] 是可以省略的，语法为：

```
数组元素类型[][]  数组名称＝{
    new 数组元素类型[]{元素值 1,元素值 2,…元素值 m1},
    new 数组元素类型[]{元素值 1,元素值 2,…元素值 m2},
    …,
    new 数组元素类型[]{元素值 1,元素值 2,…元素值 mn}
};
```

则数组元素中的值即对应直接指定的值。例如：

```
int[][] jaggedArray3＝
{
    new int[] {1,3,5,7,9},
    new int[] {0,2,4,6},
    new int[] {11,22}
};
```

需要注意的是，这种直接赋值的方法只能在声明的同时进行，同时初始化每个元素时的 new 也不能够再省略。

(4) 访问交错数组中的元素。

访问交错数组中的指定元素仍然是通过数组名和下标来访问的，其语法为：

```
数组名称[i][j]
```

表示数组中第 i 个元素中的第 j 个元素。

(5) 遍历交错数组。

①使用 for 循环控制语句遍历交错数组。

使用 for 循环控制语句遍历交错数组时，也需要通过循环的嵌套来实现。外层循环遍历数组中嵌套的数组的个数，内层循环遍历子数组中的元素。一般的方法为：

```
for(int i＝0;i＜交错数组中数组的个数;i＋＋)
{
    for(int j＝0;j＜每个数组的元素个数;j＋＋)
    {
        对 数组名[i,j] 的访问代码;
```

```
    }
}
```

在上面的一般方法中，需要考虑的问题是如何在内层循环中用一个数或一个函数描述每个子数组中包含的元素的个数。

例如，下面代码可以输出 aggedArray3 中的每个元素的值，每个子数组一行，各元素之间以空格分隔。

```
for(int i=0;i< jaggedArray3.Length;i++)
{
    for(int j=0;j< jaggedArray3[i].Length;j++)
    {
        Console.Write (jaggedArray3[i][j]+" ");
    }
    Console.WriteLine();
}
```

②使用 foreach 循环控制语句遍历交错数组。

使用 foreach 循环控制语句遍历交错数组时，由于交错数组的每个元素又是数组，因此需要 foreach 循环的嵌套来实现元素的遍历。

```
foreach(元素的数据类型[]  子数组变量名 in 数组名)
{
    foreach(元素的数据类型  元素变量名 in 子数组变量名)
    {
        对 元素变量名 执行读操作的代码;
    }
}
```

【实验 4-6】打印杨辉三角形。

内容：杨辉三角形特征。

(1) 对角线上的元素值均为 1，每一个行的第一数值亦为 1，其余的数都等于它“肩上”的两数之和。

(2) 第 1 行有 1 个数，以后每一行增加一个数，第 n 行包含 n 个数。

(3) 第 n 行数字的和为 $2^{(n-1)}$。

设计：本实验将把杨辉三角形看成一个矩阵，则该矩阵每一行的元素个数不等。用一个不规则的二维数组来表示该矩阵，其中数组的行数及每行所有的元素个数由杨辉三角形的特征决定。假设在控制台输出前 7 行的杨辉三角形。

实现：

(1) 启动 Visual Studio 2010，创建控制台应用程序 Demo4-6，并保存到适当的位置。

(2) 在 Main () 函数中添加如下示例代码：

```
static void Main(string[] args)
{
    int i, j, k=7;
    int[][] Y=new int[k][];//定义二维交错数组 Y
```

```
        //初始化数组 Y
        for (i=0; i < Y.Length; i++)
        { //Y.Length 返回的是 Y 数组的长度 7
            Y[i]=new int[i+1];//创建数组 Y 的列。数组 Y 的第 i 行含有 i+1 个元素
            /* 根据杨辉三角形的特征,每一行的第一个元素值为 1,从而对数组 Y 每一行的第一个元素
进行初始化 */
            Y[i][0]=1;
            //根据杨辉三角形的特征,对角线的元素值均为 1,对数组 Y 的对角线元素进行初始化
            Y[i][i]=1;
        }
        for (i=2; i < Y.Length; i++)
        {
            for (j=1; j < Y[i].Length - 1; j++)
            {/* Y[i].Length 是 Y[i]这个数组的长度. 根据杨辉三角形的特征,除对角线和每一行第一个
元素外,其余的数都等于它"肩上"的两数相加之和 */
                Y[i][j]=Y[i - 1][j - 1]+Y[i - 1][j];
            }
        }
        //打印杨辉三角形
        for (i=0; i < Y.Length; i++)
        {
            for (j=0; j < (Y.Length - Y[i].Length) ; j++)
            {//输出空格
                Console.Write(" ");
            }

            for (j=0; j < Y[i].Length; j++)
            {//输出数值
                Console.Write("{0} ", Y[i][j]);
            }
            Console.WriteLine();
        }
    }
```

(3) 使用 Ctrl+F5 启动程序，得到如图 4-9 所示的结果。

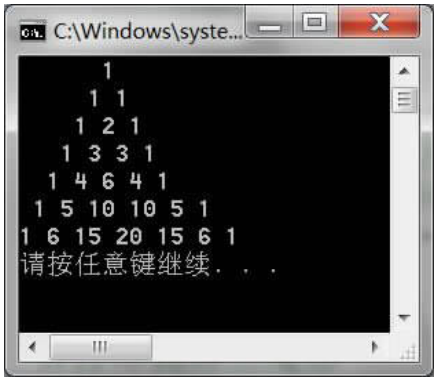

图 4-9 7 行的杨辉三角形

4.6　本章小结

本章通过“游戏平台积分排行榜”算法的实现，介绍了C＃语言中的一些基本的语法知识，包括类型转换、数组等内容。通过本章的学习，读者可以掌握以下内容。

隐式类型转换；

显式类型转换；

使用 Convert 转换；

一维数组的声明、定义与遍历；

二维数组的声明、定义与遍历。

4.7　目标达成度检验（教学效果测评）

1. 填空题

（1）类型转换的基本方法有_____、_____、_____。

（2）有 n 个元素的数组，其数组元素的序号是从_____到_____。

（3）创建数组变量一般使用关键字_____。

（4）交错数组可以认为是“_____的数组”。

（5）遍历数组元素的方式可用循环或_____结构完成。

2. 选择题

（1）有定义语句：int [,] a＝new int [5，6];，则下列正确的数组元素的引用是(　　)。

A. a（3，4）　　B. a [5] [4]

C. a [3] [4]　　D. [3，4]

（2）下列的数组定义语句，不正确的是（　　）。

A. int a [] ＝new int [5] {1，2，3，4，5};　　B. int [,] a＝new int a [3] [4];

C. int [] [] a＝new int [3] [];　　D. int [] a＝ {1，2，3，4};

3. 程序设计题

（1）从键盘输入一个正整数，按数字的相反顺序输出。

（2）从键盘输入一组整数，范围 0～100。在控制台输出这一组数中的最大数、最小数、所有数的和、所有数的平均值。

（3）从键盘上输入两个整数，由用户回答它们的和、差、积、商和取余运算结果，并统计出正确答案的个数。

（4）产生一个包含 100 个元素的 int 数组，在数组中从前到后放入 100～1，输出这个数组中的每一个元素。