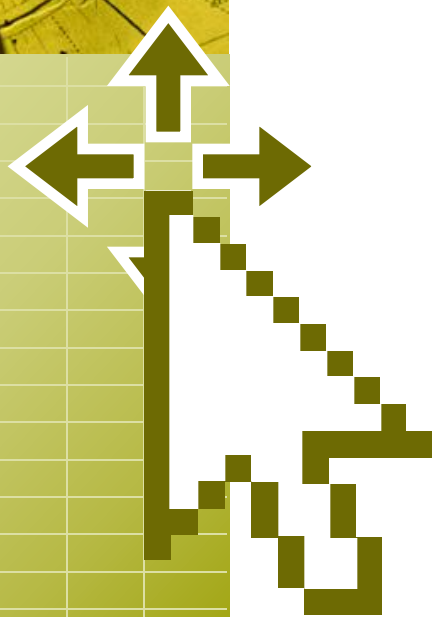




Visual C#.NET程序 设计教程（第二版）



第二章 C#程序设计基础

1 总体要求

- ❖ 掌握常量和变量概念，掌握变量的声明、初始化方法
- ❖ 掌握**C#**的常用的简单数据类型，了解枚举型、结构型，理解数据类型转换
- ❖ 掌握**C#**的运算符和表达式的概念，理解运算符运算规则，理解表达式的使用方法
- ❖ 理解数组和字符串的概念，掌握一维数组和字符串的使用方法，了解多维数组、数组型数组的应用

第二章 C#程序设计基础

2 学习重点

- ❖ C#语言中的常量、变量、数据类型、运算符、表达式等的概念
- ❖ C#语言中一维数组和字符串的概念及其使用方法

3 学习难点

- ❖ 枚举型、结构型
- ❖ 数据类型转换
- ❖ 运算符的运算规则
- ❖ 多维数组、数组型数组的概念

第二章 C#程序设计基础

主要内容

2.1 常量与变量

2.2 C#的数据类型

2.3 运算符与表达式

2.4 数组和字符串

2.1常量与变量

❖ **2.1.1 常量**

❖ **2.1.2 变量**

2.1 常量与变量

❖ C# 中常见的数据类型

数据类型	说明
int（整型）	用于存储整数，如：一天的时间是 24 小时，一月份有 31 天
double（双精度）	用于存储小数，如：早餐奶的价格 2.3 元，手机待机时间 6.5 小时
char（字符型）	用于存储单个字符，如：性别：'男'、'女'，电灯'开'、'关'
bool(布尔)	表示现实中的“真”或“假”这两个概念，分别采用 true 和 false 这两个值来表示“真”和“假”
string（字符串）	用于存储一串字符，如：“我的爱好是踢足球”，“我喜欢 C#程序”

2.1.1 常量

概念：常量是固定值，程序执行期间不会改变。

- ◆ 常量可以是任何基本数据类型，如整数常量、浮点常量、字符常量或者字符串常量、枚举常量等。
- ◆ 常量可以被当作常规变量用，只是它们的值在定义后不能被修改。

❖ 整型常量

- ◆ 整数常量可以是十进制、八进制或十六进制的常量。
前缀指定基数：**0x** 或 **0X** 表示十六进制，**0** 表示八进制，没有前缀则表示十进制。
- ◆ 整数常量也可以有后缀，可以是 **U** 和 **L** 的组合，其中 **U**和**L** 分别表示 无符号整型常量 和 长整型常量。

提示：不便停下敲代码时，可以援用网站例子。

<https://www.runoob.com/csharp/csharp-constants.html>

2.1.1 常量

❖ 浮点型常量

- ◆ 浮点型常量又分为：单精度浮点型常量和双精度型常量。单精度浮点型常量在书写时添加**f**或**F**标记，双精度型常量添加**d**或**D**标记。

❖ 小数型常量

- ◆ 小数型常量的后面必须添加**m**或**M**标记，

❖ 字符型常量

- ◆ 字符型常量，使用两个英文单引号来标记。
- ◆ C#语言还允许使用一种特殊形式的字符常量，即以反斜杠符（\）开头，后跟字符的字符序列，称之为转义字符常量，用它来表示控制及不可见的字符。

2.1.1 常量

❖ 常用的转义字符（系字符型常量）

转义符	说明
'\'	单引号'
'\"'	双引号"
'\\'	反斜线符\
'\0'	空字符
'\uhhhh'	使用十六进制形式的Unicode字符，例如字符'\u0041'表示Unicode字符A
'\a'	响铃（警报）符，与'\u0007'匹配
'\b'	退格符，与'\u0008'匹配
'\t'	Tab 符，与'\u0009'匹配。
'\r'	回车符，与'\u000D'匹配。
'\v'	垂直 Tab 符，与 '\u000B' 匹配。
'\f'	换页符，与'\u000C'匹配。
'\n'	换行符，与'\u000A'匹配。
'\0dd'	使用八进制形式的 ASCII 字符，例如字符 '\040' 表示ASCII的空格字符
'\xhh'	使用十六进制形式的ASCII 字符，例如字符'\x41'表示ASCII字符A

2.1.1 常量

❖ 布尔型常量

- ◆ 布尔型常量只有两个，一个是**true**，表示逻辑真；另一个**false**，表示逻辑假。

❖ 字符串常量

- ◆ 字符串常量，使用两个英文双引号来标记。

注意：常量的值在定义后不能再被修改，这也是常量存在的意义。

2.1.2 变量

❖ 变量的概念

- ◆ 在程序运行过程中，其值可以被改变的量称之为变量。

❖ 变量名

- ◆ 每个变量都必须有一个名字，即变量名。
- ◆ 变量命名应遵循标识符的命名规则，如必须以字母、下划线（_）和汉字打头，可包含字母、数字、下划线和汉字，不能包含空格，不能使用C#保留字等。

❖ 变量值

- ◆ 程序运行时，系统自动为变量分配内存单元，用来存储变量的值。在程序中，通过变量名来引用变量的值。

2.1.2 变量

❖ 变量的定义

- ◆ 使用变量之前必须先指定变量名、变量值的数据类型，该操作称为变量的定义。其一般形式为：

类型标识符 变量名1，变量名2，……；

int a,b,c; // a,b,c为整型变量

- ◆ 在定义变量时，应注意以下几点：
- ◆ 在多个相同类型的变量时，各变量名之间用逗号间隔，类型标识符与变量名之间至少用一个空格间隔；
- ◆ 最后一个变量名之后必须以“;”号结尾；
- ◆ 变量定义必须放在变量使用之前；

2.1.2 变量

❖ 变量的初始化

- ◆ 变量初始化就是指定变量的初始值。变量的初始化有两种形式。一种是在定义变量的同时初始化，另一种是先定义变量再初始化。

- ◆ 前者的一般形式为：

类型标识符 变量名1[=初值1]，变量名2[=初值2]，
...；

例如： `int a=12, b=-24, c=10;`

- ◆ 注意，C# 允许在定义变量时部分初始化。

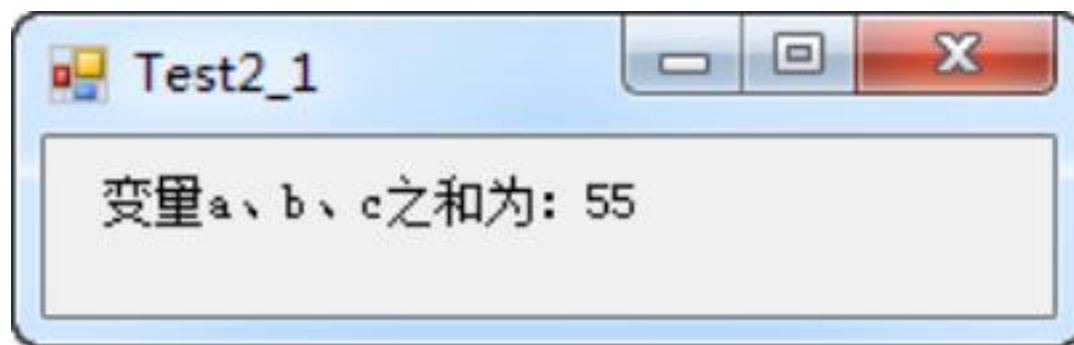
例如： `float f1=1.25, f2=3.6, f3;`

- ◆ 后者允许为多个变量设置不同的初始值，也允许为多个变量设置相同的初始值。

例如， `int a, b, c; a=1; b=2; c=3;`

2.1.2 变量

实例2-1 创建一个**Windows**应用程序，展示变量的使用方法，包括定义、初始化和引用。



2.2 C#的数据类型

- ❖ **2.2.1 简单类型（整数型、浮点型、小数型、布尔型）**
- ❖ **2.2.2 枚举型enum**
- ❖ **2.2.3 结构型struct**
- ❖ **2.2.4 数据类型转换**

2.2 C#的数据类型

- ❖ 在 **C#** 中，变量分为以下几种类型：
- ❖ 值类型 (**Value types**)
- ❖ 引用类型 (**Reference types**)
- ❖ 指针类型 (**Pointer types**)

简单类型（整数型、浮点型、小数型、布尔型）、枚举型、结构型都是值类型。值类型变量直接包含数据（简单理解，变量和其值是一体的）。比如 **int**、**char**、**float** 分别存储数字、字符、浮点数。当声明一个值类型变量时，系统会在栈上分配内存空间来存储。【记住】

2.2 C#的数据类型

- ❖ 在 **C#** 中，变量分为以下几种类型：
- ❖ 值类型 (**Value types**)
- ❖ 引用类型 (**Reference types**)
- ❖ 指针类型 (**Pointer types**)

类、接口、委托、数组、**object**、**string**都是引用类型。引用类型在做内存分配时分为两部分，引用变量本身存在栈上，引用指向的真实数据存在堆上（简单理解，变量和其值是分离的）。当栈上的引用变量销毁后，堆上的数据内存仍占用，长此以往就会造成内存泄漏，因此**.NET**设计了自动回收机制（**GC**）。【先行了解，将来记住】

补充知识：内存分成**5**个区，它们分别是堆、栈、自由存储区、全局/静态存储区和和常量存储区。

2.2.1 简单类型

❖ C#中简单类型

类型	别名	长度(位)	类型	别名	长度(位)
sbyte	System.Sbyte	8	long	System.Int64	64
byte	System.Byte	8	ulong	System.UInt64	64
char	System.Char	16	float	System.Single	32
short	System.Int16	16	double	System.Double	64
ushort	System.UInt16	16	decimal	System.Decimal	128
int	System.Int32	32	bool	System.Boolean	1
uint	System.UInt32	32			

2.2.1 简单类型

❖ 整数型

	类型	范 围	长度
sbyte	有符号字节型	-128 ~ 127	8 位
byte	字节型	0 ~ 255	8 位
char	字符型	U+0000 ~ U+FFFF （Unicode字符集中的字符）	16位
short	短整型	-32,768 ~ 32,767	16位
ushort	无符号短整型	0 ~ 65,535	16位
int	整型	-2,147,483,648 ~ 2,147,483,647	32位
uint	无符号整型	0 ~ 4,294,967,295	32位
long	长整型	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807	64位
ulong	无符号长整型	0 ~ 18,446,744,073,709,551,615	64位

2.2.1 简单类型

❖ 浮点型

- ◆ 浮点型一般用来表示一个有确定值的小数，
- ◆ float型：取值范围在 $\pm 1.5e-45$ 到 $\pm 3.4e38$ ，精度为7位
- ◆ double型：取值范围在 $\pm 5.0e-324$ 到 $\pm 1.7e308$ ，精度为15到16位

❖ 小数型decimal

- ◆ decimal型：取值范围在 $\pm 1.0 \times 10e-28$ 至 $\pm 7.9 \times 10e28$ ，精度为28到29位。

❖ 布尔型bool

- ◆ 布尔型用来表示逻辑真或逻辑假，因此只有两种取值：
true或false，

2.2.2 枚举型enum

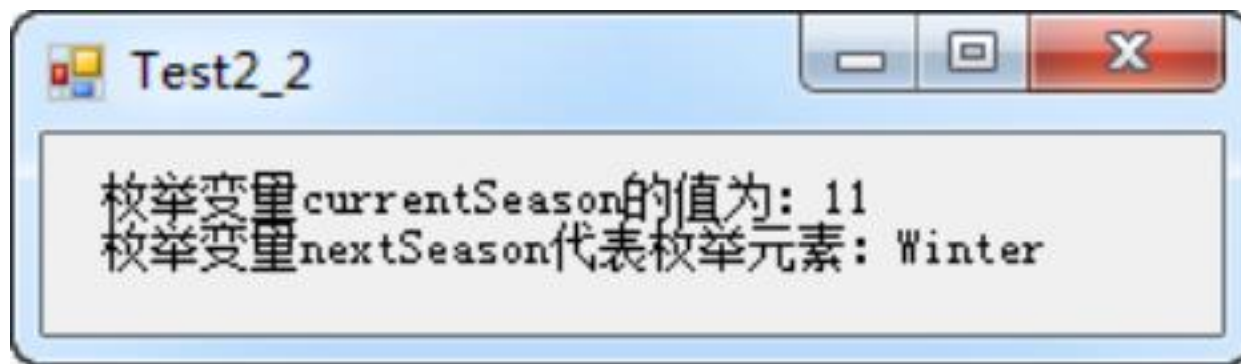
❖ 枚举型实质就是使用符号来表示的一组相互关联的数据。

`enum Months { Jan, Feb, Mar, Apr, May, Jun, Jul, Augt, Sep, Oct, Nov, Dec}。`

- ◆① 枚举元素的数据值是确定的，一旦声明就不能在程序的运行过程中更改；
- ◆② 枚举元素的个数是有限的，同样一旦声明就不能在程序的运行过程中增减；
- ◆③ 默认情况下，枚举元素的值是一个整数，第一个枚举数的值为 0，后面每个枚举数的值依次递增 1；
- ◆④ 如果需要改变默认的规则，则重写枚举元素的值即可

2.2.2 枚举型enum

【实例2-2】 创建一个**Windows**应用程序，展现枚举型的使用方法



2.2.3 结构型struct

❖ 结构型的定义

- ◆ C#的结构型必须使用**struct**来标记。C#的结构型的成员包含数据成员、方法成员等。其中，数据成员表示结构的数据项，方法成员表示对数据项的操作。一个完整的结构体示例如下：

```
struct Student
{
    public int stuNo;
    public string stuName;
    public int age;
    public char sex;
}
```

2.2.3 结构型struct

❖ 结构型的使用

- ◆ 自定义的结构型与简单类型（如int）一样，可用来定义变量。一旦定义了结构型变量，就可以通过该变量来引用其任意成员。引用结构型的成员的格式如下：

- ◆ 结构型变量.结构型成员

```
Student stu;           //定义结构型变量s  
stu.stuNo = 1001;      //为stu的成员变量stuNo赋值  
stu.stuName = "乔峰";  //为stu的成员变量stuName赋值
```


2.2.2 枚举型enum

【实例2-3】 创建一个**Windows**应用程序，展示结构型的使用方法



2.2.4 数据类型转换

❖ 隐式转换

- ◆ 隐式转换一般在不同类型的数据进行混合运算时候发生，当编译器能判断出转换的类型，而且转换不会带来精度的损失时，C#语言编译器会自动进行隐式转换。
- ◆ 隐式转换遵循以下规则：
 - ✧ 如果参与运算的数据类型不相同，则先转换成同一类型，然后进行运算；
 - ✧ 转换时按数据长度增加的方向进行，以保证精度不降低，例如int型和long型运算时，先把int数据转成long型后再进行运算；
 - ✧ 所有的浮点运算都是以双精度进行的，即使仅含float单精度量运算的表达式，也要先转换成double型，再作运算；
 - ✧ byte型和short型参与运算时，必须先转换成int型；
 - ✧ char 可以隐式转换为 ushort、int、uint、long、ulong、float、double 或 decimal，但是不存在从其他类型到 char 类型的隐式转换。

2.2.4 数据类型转换

❖ 显示转换

- ◆ 显示转换就是需要明确要求编译器完成的转换，也称强制类型转换，在转换时，需要用户明确指定转换的类型，强制类型转换的一般形式为：

(类型说明符) (待转换的数据)

- ◆ 其含义是：把待转换的数据的类型强制转换成类型说明符所表示的类型。
- ◆ 显示转换有可能造成精度损失。

2.2.4 数据类型转换

❖ 【注意】在使用强制转换时应注意以下问题：

- ◆ 待转换的数据不是单个变量时，类型说明符和特转换的数据都必须加圆括号。
- ◆ 无论是强制转换或是隐式转换，都只是为了本次运算的需要而对变量的数据长度进行的临时性转换，而不改变数据说明时对该变量定义的类型。
- ◆ C# 允许用 `System.Convert` 类提供的类型转换方法来转换数据类型，常用的转换方法有：`ToBoolean`、`ToByte`、`ToChar`、`ToInt32`、`ToSingle`、`ToString`、`ToDateTime` 等，分别表示将指定数据转换为布尔值、字节数、字符编码、整型数、单精度数、字符串、日期等

2.3 运算符与表达式

- ❖ **2.3.1** 算术运算符与表达式
- ❖ **2.3.2** 赋值运算符与表达式
- ❖ **2.3.3** 关系运算符与表达式
- ❖ **2.3.4** 逻辑运算符与表达式
- ❖ **2.3.5** 运算符优先级

2.3.1 算术运算符与表达式

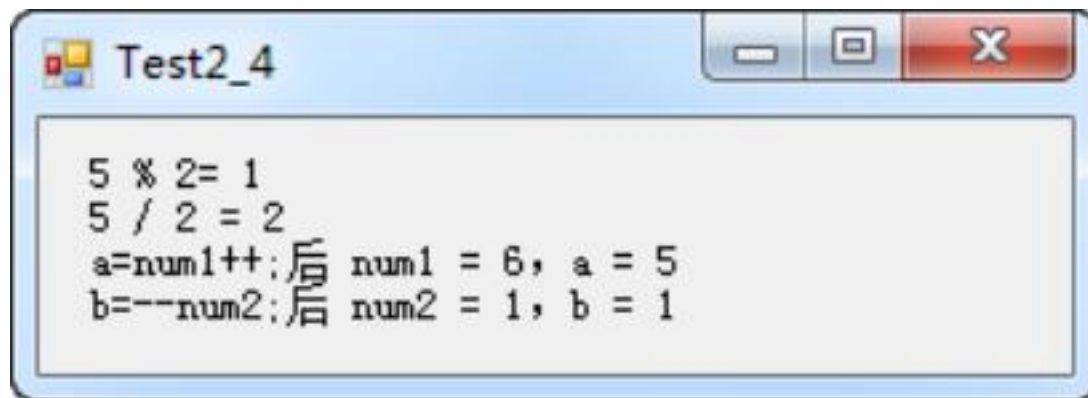
- ❖ 算术运算符用于数值运算。
- ❖ **C#**算术运算符包括**+**（加）、**-**（减）、*****（乘）、**/**（除）、**%**（求余数）、**++**（自增）、**--**（自减）共七种。
- ❖ **+**、**-**、*****、**/**、**%**是二目运算符，
- ❖ 两个整数相除的结果为整数

2.3.1 算术运算符与表达式

- ❖ **++、--**两种运算符都是单目运算符，具有右结合性（也就是优先同运算符右边的变量结合，使该变量的值增加**1**或减小**1**），而且它们的优先级比其他算术运算符高。
- ❖ 当**++**或**--**运算符置于变量的左边时，称之为前置运算，表示先进行自增或自减运算再使用变量的值
- ❖ 当**++**或**--**运算符置于变量的右边时，称之为后置运算，表示先引用变量的值再自增或自减运算。

2.3.1 算术运算符与表达式

【实例2-4】算术运算符的应用测试



```
Test2_4  
5 % 2 = 1  
5 / 2 = 2  
a=num1++;后 num1 = 6, a = 5  
b=--num2;后 num2 = 1, b = 1
```


2.3.2 赋值运算符与表达式

❖ 简单赋值运算符

- ◆ 其一般形式为：变量=表达式
- ◆ 其功能是先计算表达式的值再赋给左边的变量。赋值运算符具有右结合性。
- ◆ 【注意】在使用赋值表达式时，应注意以下两点：
- ◆ 在赋值运算中，如果赋值号两边的数据类型不同，则系统将自动先将赋值号右边的类型将转换为左边的类型再赋值；
- ◆ 在赋值运算中，不能把右边数据长度更大的数值类型隐式转换并赋值给左边数据长度更小的数值类型。

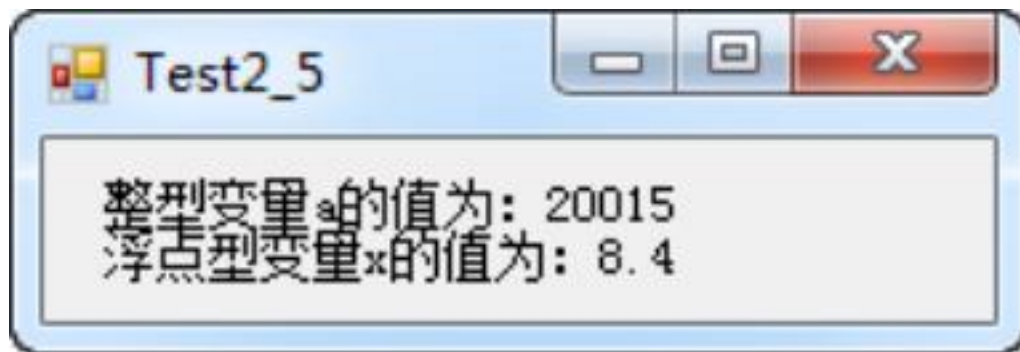
2.3.2 赋值运算符与表达式

❖ 复合赋值运算符

- ◆ 在赋值运算符“=”之前加上其它二目运算符可构成复合赋值符，常见的复合赋值运算符有： $+=$ 、 $-=$ 、 $*=$ 、 $/=$ 、 $\%=$ 等。
- ◆ 构成复合赋值表达式的一般形式为：
变量 双目运算符 = 表达式
- ◆ 它等效于
变量 = 变量 运算符 表达式

2.3.2 赋值运算符与表达式

【实例2-5】赋值运算符及隐式数据类型转换应用测试



2.3.3 关系运算符与表达式

- ❖ 关系运算符用来对两个操作数比较，以判断两个操作数之间的关系。
- ❖ **C#**的关系运算符有**==**、**!=**、**<**、**>**、**<=**、**>=**，分别是相等、不等、小于、大于、小于等于、大于等于运算。
- ❖ 关系运算符的优先级低于算术运算符。
- ❖ 由关系运算符组成的表达式称为关系表达式。关系表达式的运算结果只能是布尔型值，要么是**true**，要么是**false**。

2.3.4 逻辑运算符与表达式

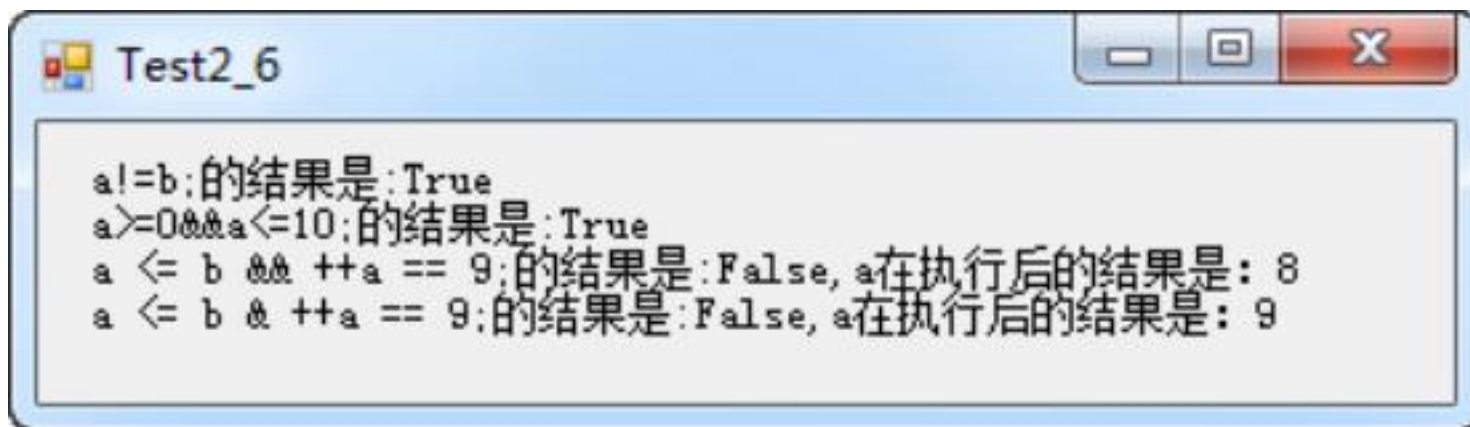
- ❖ **C#**的逻辑运算符包括**!**、**&&**或**&**、**||**或**|**、**^**，分别是逻辑“非”运算、逻辑“与”运算、逻辑“或”运算、逻辑“异或”运算。
- ❖ 逻辑运算符的优先级低于关系运算符的优先级，但高于赋值运算符的优先级。
- ❖ 由逻辑运算符组成的表达式称为逻辑表达式。逻辑表达式的运算结果只能是布尔型值，要么是**true**，要么是**false**。
- ❖ 逻辑非运算符“**!**”是，表示对某个布尔型操作数的值求反，即当操作数为**false**时运算符返回**true**。

2.3.4 逻辑运算符与表达式

- ❖ 逻辑与运算符“**&&**”或“**&**”表示对两个布尔型操作数进行与运算，当且仅当两个操作数均为 **true** 时，结果才为 **true**。运算符“**&&**”与运算符“**&**”的主要区别是，当第一个操作数为**false**时，前者不再计算第二个操作数的值。
- ❖ 逻辑或运算符“**||**”或“**|**”表示对两个布尔型操作数进行或运算，当两个操作数中只要有一个操作数为 **true** 时，结果就为 **true**。运算符“**||**”与运算符“**|**”的主要区别是，当第一个操作数为**true**时，前者不再计算第二个操作数的值。
- ❖ 逻辑异或运算符“**^**”表示对两个布尔型操作数进行异或运算，当且仅当只有一个操作数为 **true** 时，结果才为 **true**，注意或运算与异或运算的区别。

2.3.4 逻辑运算符与表达式

【实例2-6】创建一个**Windows**应用程序，测试关系运算符与逻辑运算符



```
a!=b:的结果是:True  
a>=0&&a<=10:的结果是:True  
a <= b && ++a == 9:的结果是:False, a在执行后的结果是: 8  
a <= b & ++a == 9:的结果是:False, a在执行后的结果是: 9
```

2.3.5运算符优先级

运算符	结合性
()	从左至右
++、--、!	从右至左
*/、/、%	从左至右
+、-	从左至右
<、<=、>、>=	从左至右
==、!=	从左至右
&&	从左至右
	从左至右
=、+=、*=、/=、%=、-=	从右至左

2.4 数组和字符串

- ❖ **2.4.1 一维数组**
- ❖ **2.4.2 多维数组**
- ❖ **2.4.3 数组型的数组**
- ❖ **2.4.4 字符串 `string`**

2.4.1 一维数组

- ❖ 数组是一种由若干个变量组成的集合，数组中包含的变量称为数组元素，它们具有相同的类型。
- ❖ 数组元素可以是任何类型，但没有名称，只能通过索引（又称下标，表示位置编号）来访问。
- ❖ 数组有一个“秩”，它表示和每个数组元素关联的索引的个数。数组的秩又称为数组的维度。“秩”为**1**的数组称为一维数组，“秩”大于**1**的数组称为多维数组。
- ❖ 一维数组的元素个数称为一维数组的长度。一维数组长度为**0**时，称之为空数组。一维数组的索引从零开始，具有 **n** 个元素的一维数组的索引是从 **0** 到 **n-1**。

2.4.1 一维数组

❖ 一维数组的声明和创建

- ◆ 声明和创建一维数组的一般形式如下：

数组类型[] 数组名 = new 数组类型[数组长度]

- ◆ 一维数组也可以先声明后创建。

❖ 一维数组的初始化

- ◆ 如果在声明和创建数组时没有初始化数组，则数组元素将自动初始化为该数组类型的默认初始值。
- ◆ 初始化数组有多种方式：一是在创建数组时初始化，二是先声明后初始化，三是先创建后初始化。

2.4.1 一维数组

❖ 一维数组的初始化

◆ 创建时初始化

数组类型[] 数组名 = new 数组类型[数组长度]{初始值列表}

- ✧ 其中，数组长度可省略。如果省略数组长度，系统将根据初始值的个数来确定一维数组的长度。
- ✧ 如果指定了数组长度，则C#要求初始值的个数必须和数组长度相同，
- ✧ 初始值之间以逗号作间隔。
- ✧ 创建时初始化一组数组可采用如下简写形式：
数组类型[] 数组名 = {初始值列表}

2.4.1 一维数组

❖ 一维数组的初始化

◆ 先声明后初始化

- ✘ C# 允许先声明一维数组，然后再初始化各数组元素。其一般形式如下：

数组类型[] 数组名;

数组名 = new 数组类型[数组长度]{初始值列表};

◆ 先创建后初始化

- ✘ C# 允许先声明和创建一维数组，然后逐个初始化数组元素。其一般形式如下：

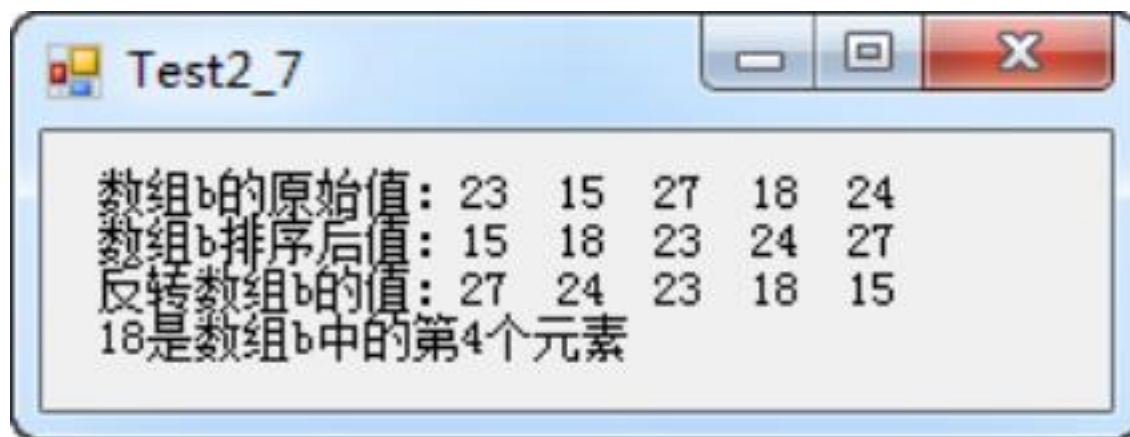
数组类型[] 数组名 = new 数组类型[数组长度];
数组元素 = 值;

2.4.1 一维数组

- ❖ 一维数组的使用
- ❖ 数组是若干个数组元素组成的。每一个数组元素相当于一个普通的变量，可以更改其值，也可以引用其值。使用数组元素的一般形式如下：数组名[索引]
- ❖ 一维数组的操作
 - ◆ C#的数组类型是从抽象基类型System.Array 派生的。
 - ◆ Array类的Length属性返回数组长度。
 - ◆ Array类的方法成员：Clear、Copy、Sort、Reverse、IndexOf、LastIndexOf、Resize等，分别用于清除数组元素的值、复制数组、对数组排序、反转数组元素的顺序、从左至右查找数组元素、从右到左查找数组元素、更改数组长度等。
 - ◆ Sort、Reverse、IndexOf、LastIndexOf、Resize只能针对一维数组进行操作。

2.4.1 一维数组

【实例2-7】数组及其应用演示



2.4.2 多维数组

❖ 多维数组的声明和创建

- ◆ 声明和创建多维数组一般形式如下：

数组类型[逗号列表] 数组名 = new 数组类型[维度长度列表]

- ◆ 逗号列表的逗号个数加1就是维度数，即如果逗号列表为一个逗号，则称为二维数组；如果为两个逗号，则称为三维数组，依此类推。维度长度列表中的每个数字定义维度的长度，数字之间以逗号作间隔。

2.4.2 多维数组

❖ 多维数组的初始化

- ◆ 以维度为单位组织初始化值，同一维度的初始值放在一对花括号{}之中。
- ◆ 可以省略维度长度列表，系统能够自动计算维度和维度的长度。但注意，逗号不能省略。
- ◆ 初始化多维数组可以使用简写格式。但如果先声明多维数组再初始化，就不能采用简写格式。
- ◆ 多维数组不允许部分初始化。

❖ 多维数组的使用

数组名[索引列表]

2.4.3 数组型的数组

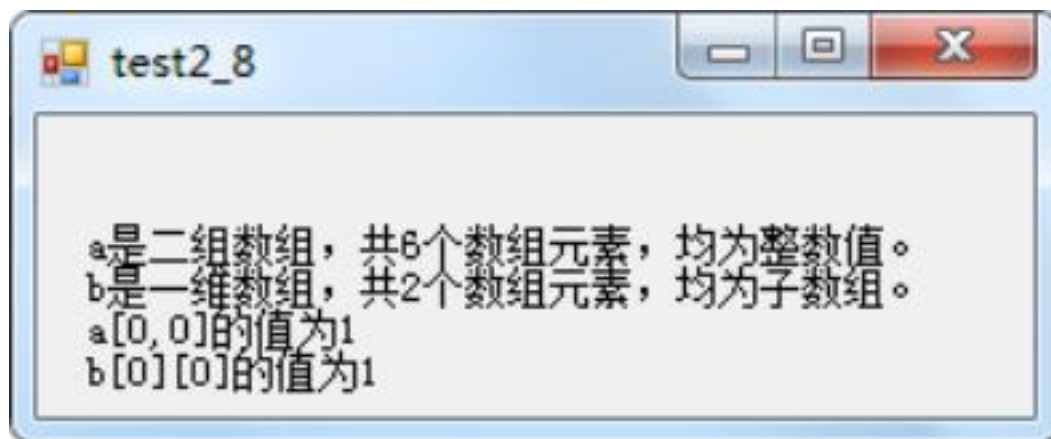
- ❖ 数组型的数组是一种由若干个数组的构成数组。
- ❖ 数组型数组的声明和创建
 - ◆ 声明数组型数组的格式如下：
数组类型[维度][子数组的维度] 数组名 = new 数组类型[维度长度][子数组的维度]
 - ◆ 其中，省略维度为一维数组，省略子数组的维度表示子数组为一维数组。

2.4.3 数组型的数组

- ❖ 数组型数组的初始化
- ❖ 数组型数组同样有多种初始化方式，包括创建时初始化、先声明后初始化等。其中，创建时初始化可省略维度长度
- ❖ 引用子数组的元素
 - ◆ 对于数组型的数组来说，可按以下格式引用子数组的每一个元素：
数组名[索引列表][索引列表]

2.3.4 逻辑运算符与表达式

【实例2-8】多维数组、数组型的数组的应用展示



2.4.4 字符串string

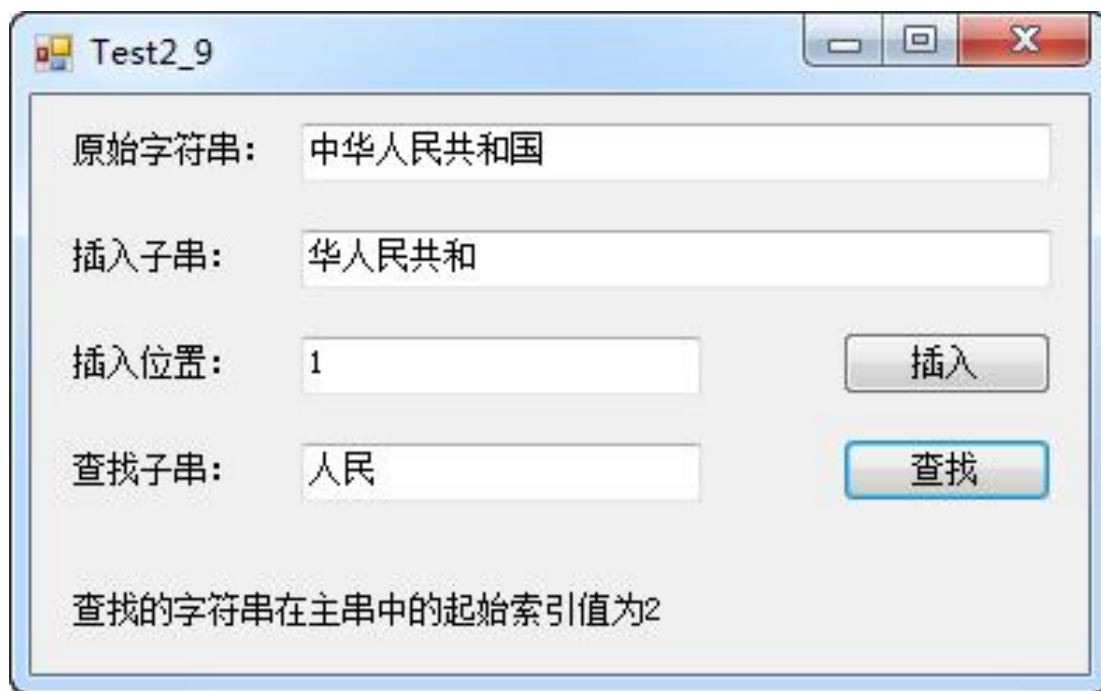
- ❖ **C# string**是一个由若干个**Unicode** 字符的组成字符数组。因此，**string**是引用型，但是很特殊（见补充），使用时就当值类型看待就好了。
- ❖ 两个字符串可以通过加号运算符（+）来连接，
- ❖ **C#**允许使用关系运算符==、!=来比较两个字符串各对应的字符是否相同。
- ❖ **C#**的字符串可以看成是一个字符数组。因此，**C#**允许通过索引来提取字符串中的字符。
- ❖ 补充：**string** 类型是特殊的引用类型，它的实例是只读的。因此，每次对**string**变量赋值时，其实都会新开辟内存赋值，原来的内存实例都会抛弃不管。因此，最终赋值行为表现出来就像值类型。

2.4.4 字符串string

- ❖ **C#的string 是 System.String 的别名。**
- ❖ **在.Net Framework之中，System.String提供的常用属性和方法有：Length、Copy、IndexOf、LastIndexOf、Insert、Remove、Replace、Split、Substring、Trim、Format等，分别用来获得字符串长度、复制字符串、从左查找字符、从右查找字符、插入字符、删除字符、替换字符、分割字符串、取子字符串、压缩字符串的空白、格式化字符串等。**
- ❖ **为了增强字符串的操作，.NET Framework.类库还提供了System.Text.StringBuilder类，可以构造可变字符串。StringBuilder类提供的常用属性和方法有：Length、Append、Insert、Remove、Replace、ToString等，分别用来获得字符串长度、追加字符、插入字符、删除字符、替换字符、将StringBuilder转化为string字符串。**

2.4.4 字符串string

【实例2-9】设计一个**Windows**应用程序，展示字符串及其应用，操作界面如图所示。



作业

- ❖ **1. 书面作业**（见本章教材）
- ❖ **2. 上机实验**（见本章教材）