2

变量与表达式

学习目标

- 了解 C#数据类型的分类
- 了解不同数据类型的转换
- 了解常量与变量的定义
- 了解运算符与表达式
- 了解控制台输入与输出

重点难点

- 不同数据类型的转换
- 常量与变量的定义
- 运算符与表达式的使用

2.1 数据类型

数据类型是程序设计的基础，在进行程序设计时，必须让计算机了解需要处理的是什么数据，以什么方式保存和显示数据。为此，C#定义了很多数据类型，对程序中声明的保存信息的变量必须说明它的数据类型。本节主要介绍常用的数据类型及其基本用法。

2.1.1 数据类型的分类

C#提供的数据类型可以分为两大类：一类是值类型，另一类是引用类型。

值类型又划分为简单类型、枚举类型和结构类型；引用类型则分为类类型、接口类型、数组类型和委托类型。

值类型和引用类型的区别在于：值类型变量直接保存变量的数据内容，当把一个值赋给一个值类型时，该值实际上被复制了；引用类型的变量保存的是数据的引用地址，引用类型的变量也叫对象，当把一个值赋给一个引用类型时，只是复制引用，实际的值仍然保留在原来的内存位置。

进行数据操作时，对于值类型，因为每个变量都保存自己的值，所以对一个变量的操作不会影响其他变量；对于引用类型，对一个变量的数据进行操作就是对这个变量在内存的数据进行操作。如果两个引用类型的变量引用同一个对象，对一个变量操作就会影响到引用同一个对象的另一个变量。

图 2-1 所示为 C#的数据类型分类。

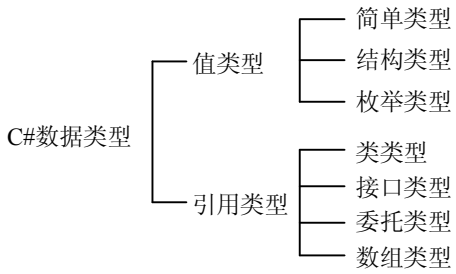

图 2-1　C#的数据类型

2.1.2　值类型

值类型是一种表示实际值的数据类型，C#的值类型包括：简单类型、枚举类型和结构类型 3 种。

1. 简单类型

C#提供已经定义好的简单类型，包括整型、实型、布尔型和字符型。

（1）整型。

整型是指数据的值是整数。根据变量在内存中所占的位数不同，C#语言提供 8 种整型。分别是：短字节型（sbyte）、字节型（byte）、短整型（short）、无符号短整型（ushort）、整型（int）、无符号整型（uint）、长整型（long）、无符号长整型（ulong）。这些不同的整型用来存储不同范围的数值，占用不同的内存空间，如表 2-1 所示。

表 2-1 中的类型指定符用于当赋值为常数时放在常数后指定类型，使用时大小写都可以。如果在给变量赋常数值时没有使用类型指定符，则默认将常数类型值隐式转换为变量类型进行赋值。

例如：

```
long x = 1234;      //int 类型的值 1234 将转换为 long 类型
```

所以可以使用“long”类型的指定符“L”，如：

```
long x = 1234L;
```

表 2-1　整型类型表示及取值范围

数据类型	取值范围	说明	类型指定符
sbyte	-128～127	1 字节有符号整数	
byte	0～255	1 字节无符号整数	
short	-32768～32767	2 字节有符号整数	
ushort	0～65535	2 字节无符号整数	
int	-2147483648～2147483647	4 字节有符号整数	
uint	0～4294967295	4 字节无符号整数	U
long	-9223372036854775808～9223372036854775807	8 字节有符号整数	L
ulong	0～18446744073709551615	8 字节无符号整数	UL

（2）实型。

实数类型是一种同时使用整数部分和小数部分表示数值的类型，分为单精度（float）、双精度（double）和小数型（decimal），它们的区别在于取值范围和精度不同。计算机对实数的运算能力大大低于整数，单精度类型常用于对精度要求不高的计算，对结果要求精确的应使用双精度类型，decimal 类型适合用于财务和货币计算。具体描述如表 2-2 所示。

表 2-2　实型类型表示及取值范围

数据类型	取值范围	说明	类型指定符
float	1.5×10^{-45}～3.4×10^{38}	4 字节单精度实数	F
double	5.0×10^{-324}～1.7×10^{308}	8 字节双精度实数	D
decimal	1.0×10^{-28}～7.9×10^{28}	16 字节实数	M

实数都是有符号类型，可以使用类型指定符表示数值的类型，大小写皆可，例如：

```
138f        //表示 float 类型的数值 138.0
32.7m       //表示 decimal 类型的数值 32.7
17.63d      //表示 double 类型的数值 17.63
```

（3）布尔型。

布尔型（bool 数据类型）是只有 true 和 false 两个无符号值的类型。如果变量只能表示“是/否”或“真/假”信息的，则将它定义为 bool 类型；可以将布尔型值赋值给布尔型变量，例如：

```
bool mybool = false;
bool b = (a > 0 && a < 10);
bool var = (20 < 30);
bool var = (a= =b);
```

（4）字符型。

字符类型用 char 表示，为单个 Unicode 字符，一个 Unicode 字符是 2 个字节长度；字符

包括数字字符、英文字符、表达符号等，字符型的变量可以用单引号引起来的字符常量直接赋值；不包含任何字符的字符串，称为空字符串。

例如：

```
char   mychar='abc';
char   mychar='A';
char   mychar='';
```

C#中提供转义符，用来表示单引号和反斜杠等特殊的字符常数，在需要使用这些特殊常数的位置，使用这些转义符来代替字符。常用转义符如表 2-3 所示。

表 2-3　常用转义符

转义符	字符名称
\'	单引号
\"	双引号
\\	反斜杠
\0	空字符
\a	发出一声响铃
\b	退格
\r	回车
\n	换行

2. 结构类型

程序设计中，往往需要用一些复杂的数据类型来表示数据。C#中使用结构类型（Structure）把多个不同类型的数据组合到一起。结构类型可以定义常数、字段、属性、方法、索引器、操作符和嵌套类型，使用结构类型的优点是节省内存空间，可以不使用继承或引用标示。

结构体是一种复合的数据类型，它允许用其他数据类型构成一个结构类型，而一个结构类型变量内的所有数据可以作为一个整体进行处理。

定义结构类型的语法如下：

```
struct      结构体标识符
{
        public   类型   成员变量名 1;
        public   类型   成员变量名 2;
        public   类型   成员变量名 3;
        …….
};
```

例如：定义一个用于表示学生的结构。

```
struct    student                    //定义学生结构体
{
        public   string   name;       //学生姓名
        public   int   chinese;       //语文成绩
```

```
        public   int   math;          //数学成绩
        public   int   eng;           //英语成绩
        public   int   aver;          //平均成绩
        public   int   count;         //总成绩
    }
```

定义好结构体变量后，就可以对结构体变量进行引用。引用结构体变量时通常是对结构体变量中的各个成员分别引用，对结构体的引用，大部分的操作也是通过引用结构体内的成员来完成，结构成员的引用方式是通过“.”运算符来进行。如下所示：

```
结构变量.成员名;
```

例如：

```
student    stu;
stu.name="张三";
stu.math=78;
int m=stu.math;
```

3. 枚举类型

枚举是这样的一种数据类型：它的值有固定的范围，在逻辑上是密不可分的整数值，这些值可以用有限个常量来叙述，例如：星期、月份等。枚举类型使用关键字 enum 定义，语法如下：

```
enum    枚举名
{
    标识符 1[=整型常数] ,
    标识符 2[=整型常数] ,
    标识符 3[=整型常数] ,
    标识符 4[=整型常数] ,
    ……..
}
```

例如：

```
enum    weekday
{
   Sun,Mon,Tue,Wed,Thu,Fri,Sat
}
```

在此枚举中，标识符在默认状态下 Sun 的值为 0，Mon 的值为 1，Tue 的值为 2，依此类推；也可以直接给标识符赋值改变元素的值，如下所示：

```
enum    weekday
{
   Sun=7,Mon=1,Tue,Wed,Thu,Fri,Sat
}
```

这样，Sun 的值为 7，Mon 的值为 1，Tue 的值为 2，依次递增。

2.1.3 引用类型

引用类型又称对象，引用类型是不直接存储变量值的，存储的是引用值的地址。使用时，引用也可以为 null，这表示当前不引用或不指向任何对象。C#的引用类型有类、数组、委托和接口 4 种。这里简单介绍 object（对象类型）和 string（字符串类型）两个类。

1. object 类

object 类是 C#所有类的基类，其他类都是由 object 类直接或间接派生来的，所以，对任何一个 object 变量，可以赋任何类型的值。例如：

```
int    a=34;                    //定义整型 a 并赋值
double    num=3.18d;            //定义实型 num 并赋值
object    obj1,obj2;            //定义 object 变量 obj1 和 obj2
obj1=a;                         //将 a 的值赋给 object 变量 obj1
obj2=num;                       //将 num 的值赋给 object 变量 obj2
```

2. string 类

string 类是专门对字符串进行操作的类。字符串是被双引号包含的若干字符。例如：

```
string    str1= "你好";         //定义字符串变量 str1 并赋值
string    str2= "C#";           //定义字符串变量 str2 并赋值
string    str3= str1+ str2;     //定义字符串变量 str3，str3 结果为“你好 C#”
char    ch=str3[0] ;            //定义字符变量 ch，ch 结果为“你”
```

2.1.4 类型转换

在 C#中，有些数据类型可以转换为另一种数据类型。

如果是一种值类型转换为另一种值类型，或者一种引用类型转换为另一种引用类型，比较常用的转换方式是：隐式转换与显式转换。对于不同值类型之间的转换，可以使用 Convert 类提供的静态方法。

1. 隐式转换

隐式转换就是将低精度数值转换为高精度数值，转换顺序如表 2-4 所示。若两种变量类型是兼容的或目标类型的取值范围大于源类型，这样的类型转换就由系统自动完成。例如：

```
int     a=100;        //定义整型变量 a 并赋值
float    b=a;         //整型变量 a 隐式转换为浮点数赋值给 b
```

表 2-4 隐式转换源类型与目标类型对应表

源类型	可以转换的目标类型
sbyte	short、int、long、float、double、decimal
byte	short、ushort、int、uint、long、float、double、decimal
char	ushort、int、uint、long、float、double、decimal
int	long、float、double、decimal
uint	long、ulong、float、double、decimal
short	int、long、float、double、decimal
ushort	int、uint、long、float、double、decimal
long	float、double、decimal
ulong	float、double、decimal
float	double

2. 显式（强制）转换

在不满足隐式转换的情况下，必须使用显式（强制）转换。显式（强制）转换是一种指令，该指令通过编译器强制将一种类型转换为另一种类型。语法为：

```
（目标类型）变量或表达式
```

例如：

```
int a=65;                  //定义整型变量 a 并赋值
short b=(short)a;          //将 a 强制转换为短整型
char  c=(char)a;           //将 a 强制转换为字符型
```

3. Convert 类

C#的 Convert 类提供多种方法来进行显式（强制）转换，语法为：

```
目标类型=Convert.转换方法(源类型)
```

例如：

```
int  num;                          //定义整型变量 num
num= Convert.ToInt16("78");        //将字符串"78"强制转换为整型并赋值给 num
```

4. ToString()方法

ToString()方法主要用于将一些类型的变量转换为字符串类型。语法为：

```
变量名. ToString()
```

例如：

```
int  a=100;
string  s=a. ToString();           //字符串变量 s 的值为"100"
```

5. Parse()方法

Parse()方法用于将特定格式的字符串转换为数值类型。语法格式为：

```
数值类型名. Parse(字符串表达式)
```

但是，语法格式中“字符串表达式”的值必须符合“数值类型名”对应的数值格式要求。例如：

```
int  x=int. Parse("123");          //将"123"转换为整型赋值给 x
int  y=int. Parse("123.4");        //因为"123.4"不符合整型的格式，所以转换失败
int  y=float. Parse("123.4");      //因为"123.4"符合浮点型的格式，所以转换成功
```

2.1.5 装箱与拆箱

对于值类型与 Object 类型之间的转换，可以用装箱和拆箱技术来实现。

C#的类型系统是统一的。它使值类型可以被看成是对象，装箱转换允许将值类型隐式转换为引用类型。装箱过程是：首先分配一个对象实例，然后将值类型的值复制到实例中。拆箱过程是：首先检查对象实例是否为给定值类型一个装了箱的值，然后将该值从实例中复制出来。

1. 装箱

装箱操作将值类型隐式转换为 Object 类型，为装箱数值分配对象实例。例如：

```
int   i=123;
object   ob=i;       //装箱
```

上述语句执行结果是在堆栈中创建对象 ob，该对象引用了堆中的 int 类型，该数值是赋给变量 i 的数值备份。

2. 拆箱

拆箱操作是将 object 类型显式转换为值类型。下面语句演示了装箱和拆箱。

```
int   i=123;
object   ob=i;       //装箱
int   j=(int)ob;     //拆箱
```

可以看出，拆箱是装箱的逆过程，要注意的是：装箱和拆箱需要遵守类型兼容的原则。

2.2 变量与常量

知道了 C#有哪些数据类型以及如何将一种数据类型转换为另一种数据类型，我们还需要知道 C#中有哪些常量和变量以及如何定义常量和变量。

2.2.1 变量

变量用来表示一个数值、一个字符串值或者一个类的对象。变量存储的值可能会发生更改，但变量名保持不变。

1. 变量声明

C#中，变量必须先声明后使用。声明变量的语法为：

```
类型标识符   变量名;
```

或

```
类型标识符   变量名 1，变量名 2，变量名 3…;
```

例如，下面语句声明了一些变量：

```
int      num;                    //声明整型变量 num
float    sum;                    //声明浮点型变量 sum
char     ch;                     //声明字符型变量 ch
string   strName,strPassword;    //声明字符串型变量 strName 和 strPassword
```

C#中，可以在声明变量的同时给变量赋值，格式为：

```
类型标识符   变量名=表达式;
```

例如：

```
double   a=0.123;                //声明双精度变量 a，同时赋值为 0.123
bool   b1=true，D2=false;        //声明两个布尔型变量，同时赋值
```

2. 变量命名规则

在变量声明时，变量名的命名应该遵守一些基本规则：

（1）变量名长度不能超过 255 个字符。

（2）变量名在有效的程序范围内是唯一的。

（3）变量名不能是关键字（保留字），但可以嵌入使用关键字。

（4）变量名用字母开头，不要使用数字开头。

关键字是一种在高级程序设计语言中属于语言成分的特殊标识符，用户是不能使用这些标识符定义各种名称的。C#中关键字如表 2-5 所示。

表 2-5　C#的关键字

abstract	base	bool	break	byte	const
continue	decimal	default	delegate	case	catch
char	checked	class	do	double	else
enum	event	explicit	extern	false	finally
fixed	float	for	foreach	goto	if
implicit	in	int	interface	internal	is
lock	long	namespace	new	null	object
operator	out	override	params	private	public
readonly	ref	return	sbyte	sealed	short
sizeof	static	string	struct	this	throw
true	try	typeof	uint	ulong	unchecked
unsafe	ushort	using	void	while	

C#中关键字都是小写字符，而程序代码编写是区分大小写的。

下面的变量名是错误的：

```
123name
class
student-name
```


除了按照基本的命名规则外，实际应用时还应注意：

1）变量名应该具有实际含义，如 studentage、sum。

2）注意大小写的不同，如 MyName、myName、myname 分别是 3 个不同的变量名。

3）采用 PascalCase 或 camelCase 方式命名变量名。

2.2.2　变量的种类

C#定义了 7 种类型的变量，即静态变量、实例变量、数组元素、值参数、引用参数、输出参数和局部变量。

```
class  A
{
    public  static  int  x;
    int  y;
    void  F(int[] v，int  a，ref  int b，out  int c)
    {
```

```
            int  i=1;
            c=a+b++;
        }
    }
```

在这段代码中，x 是静态变量，y 是实例变量，v[0]是数组元素，a 是值参数，b 是引用参数，c 是输出参数，i 是局部变量。

2.2.3 常量

常量就是值在程序的整个生命周期内始终不变的量，使用关键字 const 声明。在使用中，不可以对常量进行赋值。常量声明的语法为：

```
[访问修饰符]const[类型标识符]  常量名=值;
```

例如：

```
public  const  double  PI=3.1415926;
const    int  dayinyear=365;
```

语法中方括号[]表示其中的内容是可选项，如果采用逻辑符号“|”则表示“|”两端的内容是或者的关系。

2.3 运算符与表达式

运算符是指在表达式中执行哪些操作的符号。表达式由常量、变量、对象以及各种运算符组成。

2.3.1 运算符分类

按照操作数分类，C#语言提供 3 类运算符。包括：

一元运算符：带一个操作数的运算符称为“一元”运算符，如 i++。

二元运算符：带两个操作数的运算符称为“二元”运算符，如 x+y。

三元运算符：带三个操作数的运算符称为“三元”运算符，C#中只有一个三元运算符，即条件运算符“?:”。

表 2-6 列出了各种运算符的使用和结果

表 2-6 运算符及说明

运算	操作符	说明	示例	结果
算术运算	+	加	23+5	28
	-	减	23-5	18
	*	乘	23*5	115
	/	除	25/5	5
	%	求余	21%2	1

续表

运算	操作符	说明	示例	结果
赋值运算	=	赋值	A=8+3	A=11
连接运算	+	字符串连接	"学习"+"C#"	"学习 C#"
关系运算	==	等于	a=5，b=3；a==b	False
	>	大于	a=5，b=3；a>b	true
	<	小于	a=5，b=3；a<b	false
	>=	大于或等于	a=5，b=3；a>=b	true
	<=	小于或等于	a=5，b=3；a<=b	false
	!=	不等于	a=5，b=3；a!=b	true
逻辑运算	!	逻辑非	!A	A 为 true，结果为 false，否则为 true
	&&	逻辑与	A&&B	A 与 B 都为 true，结果为 true，否则为 false
	||	逻辑或	A||B	A 与 B 都为 false，结果为 false，否则为 true

2.3.2 算术运算符

C#中算术运算符就是用于常规运算功能的符号，包括加法运算符（+）、减法运算符（-）、乘法运算符（*）、除法运算符（/）、求余运算符（%）、自增运算符（++）、自减运算符（--）。

要注意的是，乘法、除法运算符只适用于整数和实数之间的操作；默认运算返回值与精度高的类型相同。求余运算符用来求除法的余数。例如：

```
int     a=3/2;        //结果为 1
double  a=3.0/2;      //结果为 1.5
```

【例 2-1】变量自增运算。

```
using System;
namespace selfplus
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 3;
            int b = a++;                //先使用后自增
            Console.WriteLine("b={0}", b);
            b = ++a;                    //先自增后使用
            Console.WriteLine("b={0}", b);
            Console.Read();
        }
    }
}
```

程序中，b 变量第一次先使用后自增，所以 b=2，a=3；第二次先自增后使用，所以 b=4，a=4。自增、自减运算只能用于变量，不能用于表达式或常量。

2.3.3 赋值运算符

赋值运算符用于改变变量的值，即为变量赋值。C#中提供了一个简单赋值运算符“=”和多个复合赋值运算符。包括：+=、-=、*=、/=、%=、<<=、>>=、&=、^=和|=。

赋值运算符是左结合，将右边的操作数的值赋给左边的操作数，左操作数必须是变量。C#也可以对变量进行连续赋值，此时赋值运算从右向左进行。例如：

```
int a, b, c;
a = 5;
b = c = a;
```

程序先将数值 5 赋值给变量 a，再将 a 的值 5 赋值给变量 c，最后将变量 c 的值赋值给变量 b，所以 a、b、c 的值都为 5。

复合赋值运算符是将一个其他运算符加上简单赋值运算符，含义为：将左操作数和右操作数按其他运算符功能进行运算，将结果的值赋值给左操作数。如：

```
x += 10; //等价于 x=x+10
y *= 2; //等价于 y=y*2
```

如果赋值运算两边操作数类型不一致，需要先进行类型转换。

2.3.4 关系运算符

关系运算符用于在程序中比较两个值大小，关系运算的结果是布尔型。关系运算符包括：==、!=、<、>、<=、>=。例如：

```
int a = 100, b = 80;
bool bl = a > b;
Console.WriteLine("结果为：{0}", bl);   //结果为：True
```

如果关系运算符两边是字符串，则比较两个字符串的大小，比较方法是依次按照字符串字符在 ASCII 表中的大小比较。

2.3.5 逻辑运算符

逻辑运算中，使用逻辑运算符将运算对象连接起来形成逻辑表达式，逻辑运算只有两个结果：true 和 false，表示真和假。例如：

```
bool b1 = !true; // b1 为：false
bool b2 = 6 > 3 && 1 > 2; //b2 为：false
bool b3 = 5 > 3 || 2 > 4; //b3 为：true
```

2.3.6 三目运算符

?:运算符是根据布尔表达式的值返回两个值中的一个。如果条件为真，则返回第一个值；否则，返回第二个值。语法为：

（布尔表达式）?值 1:值 2

【例 2-2】比较两数大小。

```
using System;
namespace smys
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 23;
            int b = 19;
             string   ss=(a>b)?"a 大于 b ":" a 等于或小于 b ";//三目运算符
            Console.WriteLine(ss);
           Console.Read();
        }
    }
}
```

2.3.7 运算符优先级

当一个表达式包含多个运算符时，表达式的值就由各运算符的优先级决定。表 2-7 为运算符的优先级描述。

表 2-7 运算符优先级（由高到低）

特殊运算符	()
一元运算符	+（正）、-（负）、!（逻辑非）
乘/除运算符	*、/、%
加/减运算符	+、-
移位运算符	<<、>>
关系运算符	<、>、<=、>=、is
比较运算符	==、!=
逻辑与运算符	&
逻辑异或运算符	^
逻辑或运算符	|
条件与运算符	&&
条件或运算符	||
三目运算符	?:
赋值运算符	=、+=、-=、*=、/=、%=

2.4 输入与输出

通常编写的程序都需要实现一种交互：程序接收一定的数据输入，并对输入的数据进行处理，最后将处理的结果反馈给用户，也就是输出。一般情况下，数据的输入和输出方式有两种：从控制台输入，或从文件中输入；输出到控制台，或输出到文件中。这里主要介绍控制台的输入和输出，文件系统的输入和输出将在后面的章节介绍。

控制台输入/输出主要通过命名空间 System 中的类 Console 来实现。该类表示标准的输入/输出流和错误流，提供了从控制台读写字符的基本功能。控制台输入主要通过 Console 类的 Read 和 ReadLine 方法来实现，控制台输出主要通过 Console 类的 Write 和 WriteLine 方法来实现。

1. Console.WriteLine()方法

WriteLine()方法的作用是将信息输出到控制台，并在信息的后面添加一个回车换行符来产生一个新行。

在 WriteLine()方法中，可以应用格式化字符串方式向控制台输出，格式为：

```
Console.WriteLine("格式字符串",变量列表);
```

【例 2-3】用 WriteLine()方法输出变量值。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Hello
{
    class Program
    {
        static void Main(string[] args)
        {
            string name="丽莎";
            int score=95;
            Console.WriteLine("姓名：{0} ，成绩：{1}", name,score);
            Console.ReadLine();//等待用户输入，直到敲回车 DOS 窗口才关闭

        }
    }
}
```

其中，格式字符串{0}、{1}叫做占位符，它占的就是后面的 name、score 变量的位置。

在格式字符串中，我们依次使用{0}，{1}，{2}代表要输出的变量，然后将变量依次排列在变量列表中，0 对应变量列表的第 1 个变量，1 对应变量列表的第 2 个变量，依此类推。

程序运行结果如图 2-2 所示。

图 2-2　例 2-3 运行结果

2. Console.Write()方法

Write()方法和 WriteLine()方法类似，都是将信息输出到控制台，但 Write()方法输出信息到屏幕后并不产生一个新行，即换行符不会连同输出信息一起输出到屏幕上，光标将停留在所输出信息的末尾。

【例 2-4】用 Write()方法输出变量值。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Hello
{
    class Program
    {
        static void Main(string[] args)
        {
            string name="丽莎";
            int score=95;
            Console.Write ("姓名：{0} ", name);
            Console.Write ("\n");//输出回车换行符
            Console.Write ("成绩：{0}", score);
            Console.ReadLine();//等待用户输入，直到敲回车 DOS 窗口才关闭
        }
    }
}
```

程序运行结果如图 2-3 所示。

图 2-3　例 2-4 运行结果

3. Console.ReadLine()方法

ReadLine()方法用来从控制台读取一行数据，一次读取一行字符的输入，并且直到用户按

下 Enter 键才会返回，但 ReadLine()方法并不接收 Enter 键。如果 ReadLine()方法没有接收到任何输入，或者接收了无效的输入，那么 ReadLine()方法将返回 null。

【例 2-5】用 ReadLine()方法接收用户输入，然后输出。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Hello
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("请输入您的姓名：");
            string name= Console.ReadLine();
            Console.WriteLine("请输入您的兴趣爱好：");
            string hobby= Console.ReadLine();
            Console.WriteLine("{0}的兴趣爱好是：{1}", name,hobby);
            Console.ReadLine();//使程序执行不会自动退出调试环境
        }
    }
}
```

程序运行结果如图 2-4 所示。

图 2-4　例 2-5 运行结果

4. Console.Read ()方法

Read()方法的作用是从输入流（控制台）读取下一个字符，Read()方法一次只能从输入流读取一个字符，并且直到用户按 Enter 键才会返回。当这个方法返回时，如果输入流中包含有效的输入，则返回该字符 Unicode 编码对应的十进制值；如果输入流中没有数据，则返回-1。

如果用户输入了多个字符，则 Read()方法只返回用户输入的第 1 个字符，但是，用户可以通过多次调用 Read()方法来获取所有输入的字符。

【例 2-6】用 Read()方法接收用户输入，然后显示接收的内容。

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
namespace Hello
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("请输入字符：");
            int s= Console.Read ();
            Console.WriteLine("您刚才输入的内容为：{0}", s);
            Console.ReadKey();//DOS 窗口接收一个字符再退出
          }
    }
}
```

程序运行结果如图 2-5 所示。这里，97 是字母 a 的 Unicode 编码对应的十进制值。

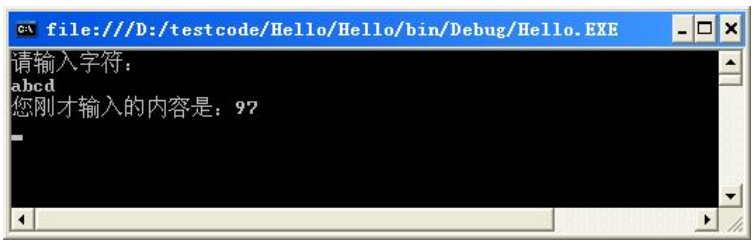

图 2-5　例 2-6 运行结果

2.5　习题

1．C#中有哪些基本数据类型？
2．变量和常量的概念及区别是什么？
3．C#中关键字是什么？
4．值类型变量与引用类型变量的区别是什么？它们之间如何转换？
5．表达式是什么？
6．静态变量有什么特点？
7．Read()和 ReadLine()的区别是什么？
8．Write()和 WriteLine()的区别是什么？