عنوان مضمون

Visual Programming-I

توسط : صفری

خزان1397

operators

operators

- C# contains a number of ***operators*** for this purpose.
- By combining operators with variables and literal values (together referred to as *operands* when used with operators), you can create ***expressions***, which are the basic building blocks of computation.

- Operators can be roughly classified into three categories:
  - ➤ **Unary** — Act on single operands
  - ➤ **Binary** — Act on two operands
  - ➤ **Ternary** — Act on three operands

Mathematical Operators

TABLE of Simple Mathematical Operators

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
+	Binary	`var1 = var2 + var3;`	`var1` is assigned the value that is the sum of `var2` and `var3`.
-	Binary	`var1 = var2 - var3;`	`var1` is assigned the value that is the value of `var3` subtracted from the value of `var2`.
*	Binary	`var1 = var2 * var3;`	`var1` is assigned the value that is the product of `var2` and `var3`.
/	Binary	`var1 = var2 / var3;`	`var1` is assigned the value that is the result of dividing `var2` by `var3`.
%	Binary	`var1 = var2 % var3;`	`var1` is assigned the value that is the remainder when `var2` is divided by `var3`.
+	Unary	`var1 = +var2;`	`var1` is assigned the value of `var2`.
-	Unary	`var1 = -var2;`	`var1` is assigned the value of `var2` multiplied by `-1`.

Mathematical Operators

- TABLE of The String Concatenation Operator

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
+	Binary	`var1 = var2 + var3;`	`var1` is assigned the value that is the concatenation of the two strings stored in `var2` and `var3`.

- None of the other mathematical operators, however, work with strings.

Mathematical Operators

- TABLE of Increment and Decrement Operators

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
`++`	Unary	`var1 = ++var2;`	`var1` is assigned the value of `var2 + 1`. `var2` is incremented by `1`.
`--`	Unary	`var1 = --var2;`	`var1` is assigned the value of `var2 - 1`. `var2` is decremented by `1`.
`++`	Unary	`var1 = var2++;`	`var1` is assigned the value of `var2`. `var2` is incremented by `1`.
`--`	Unary	`var1 = var2--;`	`var1` is assigned the value of `var2`. `var2` is decremented by `1`.

Mathematical Operators

- Placing one of these operators before its operand means that(++ a)
  the operand is affected before any other computation takes place.
- Placing it after the operand means that(a --)
  the operand is affected after all other computation of the expression is completed.
- This merits another example! Consider this code:

```
int var1, var2 = 5, var3 = 6;
var1 = var2++ * --var3;
Console.WriteLine(var1);
Console.ReadKey();
```

Assignment Operators

- So far, you've been using the simple = assignment operator, and it may come as a surprise that any other assignment operators exist at all.

OPERATOR	CATEGORY	EXAMPLE EXPRESSION	RESULT
=	Binary	`var1 = var2;`	`var1` is assigned the value of `var2`.
+=	Binary	`var1 += var2;`	`var1` is assigned the value that is the sum of `var1` and `var2`.
-=	Binary	`var1 -= var2;`	`var1` is assigned the value that is the value of `var2` subtracted from the value of `var1`.
*=	Binary	`var1 *= var2;`	`var1` is assigned the value that is the product of `var1` and `var2`.
/=	Binary	`var1 /= var2;`	`var1` is assigned the value that is the result of dividing `var1` by `var2`.
%=	Binary	`var1 %= var2;`	`var1` is assigned the value that is the remainder when `var1` is divided by `var2`.

so code like var1 += var2;

has exactly the same result as var1 = var1 + var2;

Operator Precedence

- When an expression is evaluated, each operator is processed in sequence**, but this doesn't necessarily mean evaluating these operators from left to right**. As a trivial example, consider the following:

```
var1 = var2 + var3;
```

Here, the + operator acts before the = operator.

- This example:

```
var1 = (var2 + var3) * var4;
```

- Here, the content of the parentheses is evaluated first, meaning that the + operator acts before the * operator.

Operator Precedence

- TABLE of Operator Precedence:

PRECEDENCE	OPERATORS
Highest	++, -- (used as prefixes); +, - (unary)
	*, /, %
	+, -
	=, *=, /=, %=, +=, -=
Lowest	++, -- (used as postfixes)

Namespaces

Namespaces

- Namespaces are the .NET way **of providing containers for application code**, such that code and its contents may be uniquely identified.
- The class names declared in one namespace does not conflict with the same class names declared in another.
- Namespaces are also used as a means of categorizing items in the .NET Framework. Most of these items are type definitions, such as the simple types in this chapter (System.Int32 and so on).

```
namespace namespace_name {
   // code declarations
}
```

Namespaces

- To call the namespace-enabled version of either function or variable, prepend the namespace name as follows −

```
namespace_name.item_name;
```

```
using System;
namespace first_space {
  class namespace_cl {
    public void func() {
      Console.WriteLine("Inside first_space");
    }
  }
}
namespace second_space {
  class namespace_cl {
    public void func() {
      Console.WriteLine("Inside second_space");
    }
  }
}
class TestClass {
  static void Main(string[] args) {
    first_space.namespace_cl fc = new first_space.namespace_cl();
    second_space.namespace_cl sc = new second_space.namespace_cl();
    fc.func();
    sc.func();
    Console.ReadKey();
  }
}
```

The *using* Keyword

- The using keyword states that the program is using the names in the given namespace. For example, we are using the System namespace in our programs. The class Console is defined there. We just write −

- Console.WriteLine ("Hello there");

- We could have written the fully qualified name as −

- System.Console.WriteLine("Hello there");

```
using System;
using first_space;
using second_space;
namespace first_space {
  class abc {
    public void func() {
      Console.WriteLine("Inside first_space");
    }
  }
}
namespace second_space {
  class efg {
    public void func() {
      Console.WriteLine("Inside second_space");
    }
  }
}
class TestClass {
  static void Main(string[] args) {
    abc fc = new abc();
    efg sc = new efg();
    fc.func();
    sc.func();
    Console.ReadKey();
  }
}
```