

C# Operators

By Nimesh Kumar Dagur

C# Operators

- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.
- Operators are used for building expressions in C#.
- To calculate the value of variable or performs operation in variable you will have to make proper expression.
- These expressions are made using C# operators.

C# Operators

C# provides wide range of operators:

- Arithmetic Operators
- Assignment Operators
- Unary operators
- Relational (Comparison) Operators
- Logical Operators
- Ternary operator

Arithmetic Operators

- Arithmetic Operators are used for basic mathematical calculation in C# programming.

Operator	Description	Examples
+	Add numbers	$X = \text{num1} + \text{num2}$
-	Subtract numbers	$X = \text{num1} - \text{num2}$
*	Multiply numbers	$X = \text{num1} * \text{num2}$
/	Divide numbers	$X = \text{num1} / \text{num2}$
%	Divide two numbers and returns remainder	$X = 22 \% 10$ then X will be $X = 2$

Example:

```
using System;
namespace Arithmetic_Operators
{
    class Program
    {
        static void Main(string[] args)
        {
            int num1, num2;
            int add, sub, mul;
            float div;
            Console.Write("Enter first number\t\t");
            num1 = Convert.ToInt32(Console.ReadLine());
            Console.Write("\n\nEnter second number\t\t");
            num2 = Convert.ToInt32(Console.ReadLine());
            add = num1 + num2;
            sub = num1 - num2;
            mul = num1 * num2;
            div = (float)num1 / num2;
            Console.WriteLine("\n\n=====\\n");
            Console.WriteLine("Addition\t\t{0}", add);
            Console.WriteLine("Subtraction\t\t{0}", sub);
            Console.WriteLine("Multiplication\t\t{0}", mul);
            Console.WriteLine("Division\t\t{0}", div);
            Console.WriteLine("\n\n=====\\n");
            Console.ReadLine();
        }
    }
}
```

num1=5

num2=10

C# Assignment Operators

- The C# assignment operator is generally suffix with arithmetic operators.
- The symbol of c sharp assignment operator is "=" without quotes.
- The assignment operator widely used with C# programming.

C# Assignment Operators

Assignment Operators	Usage	Examples
= (Equal to)	result=5	Assign the value 5 for result
+= (Plus Equal to)	result+=5	Same as result=result+5
-= (Minus Equal to)	result-=5	Same as result=result-5
= (Multiply Equal to)	result=5	Same as result=result*5
/= (Divide Equal to)	result/=5	Same as result=result/5
%= (Modulus Equal to)	result%=5	Same as result=result%5

Example:

```
using System;
namespace Assignment_operators
{
    class Program
    {
        static void Main(string[] args)
        {
            int num1, num2;
            num1 = 10;
            num2 = 5;
            num1 += num2; // same as num1=num1+num2
            Console.WriteLine("Add = {0}", num1);
            num1 -= num2; // same as num1=num1-num2
            Console.WriteLine("\n\nSubtraction = {0}", num1);
            num1 *= num2; // same as num1=num1*num2
            Console.WriteLine("\n\nMultiplication={0}", num1);
            num1 %= num2; // same as num1=num1%num2
            Console.WriteLine("\n\nModulus = {0}", num1);
            Console.ReadLine();
        }
    }
}
```

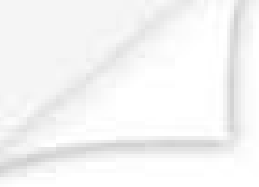

C# Unary Operators

- The C# unary operator is widely used for increment or decrement value by 1.
- This operator widely used with loop constructs to increment loop by 1.

++ Increment Operator:

- This operator is pronounced as **increment operator**.
- It is used for incrementing value by 1.
- It is used in C# programming by two types:
- **Pre-increment (++i)** and **Post-increment (i++)**.
- In pre-increment, first it increments by 1 then loop executes whereas in Post-increment, the loop executes then it increments by 1.

C# Unary Operators



```
using System;

namespace Increment_Operator
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 0; // initialization

            i++; // i incremented by one. It is post increment

            Console.WriteLine("The value of i is {0}", i);

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

C# Unary Operators

-- Decrement Operator:

- The behavior of decrement operator is just opposite from increment operator.
- It is used for decrementing the value by one.
- It has also two types:
- **Pre-Decrement (--i)** and **Post Decrement (i--)**.
- In pre-decrement the value is decremented by one then loop executes whereas in post-decrement the loop executed then the value decrements by one

C# Unary Operators

```
using System;

namespace Decrement_Operator
{
    class Program
    {
        static void Main(string[] args)
        {
            int i=5; // Initialization
            Console.WriteLine("The Value of i is {0}", i);

            i--; // i decremented by one. It is post-decrement

            Console.WriteLine("\nNow the value of i is {0}",i);

            Console.ReadLine();

        }
    }
}
```

C# Comparison Operators

- The C# comparison operator is used to compare two operands.
- It returns true or false based on comparison.

Operator	Name	Examples
<	Less than	x<5 (returns true)
>	Greater than	x>5 (returns false)
<=	Less than equal to	x<=2 (returns true)
>=	Greater than equal to	x>=2 (returns true)
==	Equal equal to	x==2 (returns true)
!=	Not equal to	x!=2 (returns false)

Example:

num1=5

num2=10

```
using System;

namespace Comparison_Operator
{
    class Program
    {
        static void Main(string[] args)
        {
            int num1, num2;

            //Accepting two inputs from the user
            Console.Write("Enter first number\t");
            num1 = Convert.ToInt32(Console.ReadLine());
            Console.Write("Enter second number\t");
            num2 = Convert.ToInt32(Console.ReadLine());

            //Processing comparison
            //Check whether num1 is greater than or not
            if (num1 > num2)
            {
                Console.WriteLine("{0} is greater than {1}",
                    num1, num2);
            }
            //Check whether num2 is greater than or not
            else if (num2 > num1)
            {
                Console.WriteLine("{0} is greater than {1}",
                    num2, num1);
            }
            else
            {
                Console.WriteLine("{0} and {1} are equal",
                    num1, num2);
            }
            Console.ReadLine();
        }
    }
}
```

C# Logical Operator

- The C# Logical Operators also evaluate the values and returns true or false as output.
- Based on true-false the program behave dynamically at run time.
- These operators are widely used with C# programming.
- **&& Operator (and Operator)**
- **|| Operator (or Operator)**
- **! Operator (not Operator)**
- **^ Operator (xor operator)**

&& Operator

- It is pronounced as **and operator**.
- It returns true if both or all the condition is true and return false if any of the condition is false.


```
using System;

namespace and_operator
{
    class Program
    {
        static void Main(string[] args)
        {
            string name, password;

            name="Steven";
            password="Steven123";

            // evaluating both expression and returns true if
            // all are true.
            if (name == "Steven" && password == "Steven123")
            {
                Console.WriteLine("Login Successful");
            }
            else
            {
                Console.WriteLine("Unauthorised access");
            }
            Console.ReadLine();
        }
    }
}
```

|| Operator

- It is pronounced as **or operator**.
- It also returns true or false based on condition.
- If any one of the condition matches then it returns true but if both or all the conditions are false then it returns false.

```
using System;

namespace Or_operator
{
    class Program
    {
        static void Main(string[] args)
        {
            string username, userpassword;

            label: //Creating label

            Console.Write("\n\nEnter your login name:\t");
            username = Console.ReadLine();

            Console.Write("\nEnter your password:\t");
            userpassword = Console.ReadLine();
```

```
if ((username == "Steven" || username == "Clark")
    && {userpassword == "Steven Clark"})
{
    Console.WriteLine("\nLogin Successful.");
}
else
{
    Console.WriteLine("\nUnauthorised Access.
    Aborting...");
}

Console.Write("\n\nPress Y or y for continue.:\t");
char ans = Convert.ToChar(Console.ReadLine());

if (ans == 'Y' || ans == 'y')
{
    goto label; //goto label
}

Console.WriteLine("Press Enter for Aborting...");
Console.ReadLine();
}

}
```

! Operator:

- It is pronounced as **not operator**.
- It returns true if expression is false.
- The following demonstration will clear the concept of not operator.

```
using System;

namespace Not_Operator
{
    class Program
    {
        static void Main(string[] args)
        {
            string username, password;

            Console.Write("Enter user name:\t");
            username = Console.ReadLine();
            Console.Write("Enter Password:\t");
            password = Console.ReadLine();

            if (!(username == "Steven" && password == "Clark"))
            {
                Console.WriteLine("\nLogin Successful");
            }
            else
            {
                Console.WriteLine("\nUnauthorised Access.
                Aborting...");
            }
            Console.ReadLine();
        }
    }
}
```

^ Operator

- It is pronounced as **xor operator**.
- It returns false if the following condition matches:
 - (i) if both or all the expression returns true.
 - (ii) If both or all the expression returns false.

```
using System;

namespace xor_operator
{
    class Program
    {
        static void Main(string[] args)
        {
            string name, password;

            name = "Steven";
            password = "Clark";

            //it returns false because both expression match.
            if ((name == "Steven") ^ (password == "Clark"))
            {
                Console.WriteLine("Access granted...");
            }
            else
            {
                Console.WriteLine("Access Denied. Aborting...");
            }
            Console.ReadLine();
        }
    }
}
```

Ternary Operator

```
using System;

class Program
{
    static void Main()
    {
        int Number = 10;

        bool IsNumber10;

        if (Number == 10)
        {
            IsNumber10 = true;
        }
        else
        {
            IsNumber10 = false;
        }


        Console.WriteLine("Number == 10 is {0}", IsNumber10);
    }
}
```

```
using System;

class Program
{
    static void Main()
    {
        int Number = 15;

        bool IsNumber10 = Number == 10 ? true : false;

        Console.WriteLine("Number == 10 is {0}", IsNumber10);
    }
}
```



Console.WriteLine("Number == 10 is {0}", IsNumber10);

C# - Operators Precedence and Associativity

Operators Precedence:

- Operator precedence determines the grouping of terms in an expression.
- This affects evaluation of an expression.
- Certain operators have higher precedence than others;
- **Example:**
 - o The multiplication operator has higher precedence than the addition operator.

$$x = 7 + 3 * 2;$$

here, x is assigned 13, not 20 because operator * has higher precedence than + .

C# - Operators Precedence and Associativity

Operator Associativity:

- When an operand occurs between two operators with the same precedence, the associativity of the operators controls the order in which the operations are performed:
- Except for the assignment operators, all binary operators are **left-associative**, meaning that operations are performed from **left to right**.

Example: $x + y + z$ is evaluated as $(x + y) + z$.


- The assignment operators and the conditional operator (?:) are **right-associative**, meaning that operations are performed from **right to left**.

Example: $x = y = z$ is evaluated as $x = (y = z)$.

- **Precedence and associativity can be controlled using parentheses**

C# - Operators Precedence and Associativity

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right



Highest precedence

Lowest precedence

Arithmetic Expression	Evaluation
<code>3 + 2 - 1</code>	<code>((3 + 2) - 1)</code>
<code>2 + 6 * 7</code>	<code>(2 + (6 * 7))</code>
<code>-5+7- -6</code>	<code>(((-5)+7) - (-6))</code>
<code>2+4/5</code>	<code>(2+(4/5))</code>
<code>13 % 5</code>	<code>(13 % 5)</code>
<code>11.5 % 2.5</code>	<code>(11.5 % 2.5)</code>
<code>10 / 0</code>	ArithmeticException
<code>2+4.0/5</code>	<code>(2.0+(4.0/5.0))</code>
<code>4.0 / 0.0</code>	<code>(4.0 / 0.0)</code>
<code>-4.0 / 0.0</code>	<code>((-4.0) / 0.0)</code>
<code>0.0 / 0.0</code>	<code>(0.0 / 0.0)</code>

C# - Operators Precedence and Associativity : Example

```
using System;
namespace OperatorsAppl
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 20;
            int b = 10;
            int c = 15;
            int d = 5;
            int e;

            e = (a + b) * c / d;    // ( 30 * 15 ) / 5
            Console.WriteLine("Value of (a + b) * c / d is : {0}", e);

            e = ((a + b) * c) / d;    // (30 * 15) / 5
            Console.WriteLine("Value of ((a + b) * c) / d is : {0}", e);

            e = (a + b) * (c / d);    // (30) * (15/5)
            Console.WriteLine("Value of (a + b) * (c / d) is : {0}", e);

            e = a + (b * c) / d;    // 20 + (150/5)
            Console.WriteLine("Value of a + (b * c) / d is : {0}", e);
            Console.ReadLine();
        }
    }
}
```

The if...else if...else Statement

- Syntax:

```
if(boolean_expression 1)
{
    /* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2)
{
    /* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3)
{
    /* Executes when the boolean expression 3 is true */
}
else
{
    /* executes when the none of the above condition is true */
}
```

```
using System;
namespace MyIfStatement
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter a vowel: ");
            char mychar = Convert.ToChar(Console.ReadLine());
            char x = Char.ToUpper(mychar);
            if (x == 'A') //first if statement
                Console.WriteLine("The entered vowel is: A");
            else if (x == 'E') //second if statement
                Console.WriteLine("The entered vowel is: E");
            else if (x == 'I') //third if statement
                Console.WriteLine("The entered vowel is: I");
            else if (x == 'O') //fourth if statement
                Console.WriteLine("The entered vowel is: O");
            else if (x == 'U') //fifth if statement
                Console.WriteLine("The entered vowel is: U");
            else
                Console.WriteLine("The entered vowel is not a Vowel");

            Console.Read();
        }
    }
}
```
