**C# Operators**

In this article, we will learn everything about different types of operators in C# programming language and how to use them.

Operators are symbols that are used to perform operations on operands. Operands may be variables and/or constants.

**For example**, in 2+3, + is an operator that is used to carry out addition operation, while 2 and 3 are operands.

Operators are used to manipulate variables and values in a program. C# supports a number of operators that are classified based on the type of operations they perform.

**1. Basic Assignment Operator**

Basic assignment operator (=) is used to assign values to variables. For example,

double x;

x = 50.05;

Here, 50.05 is assigned to x.

**Example 1: Basic Assignment Operator**

using System;

namespace Operator

{

class AssignmentOperator

{

public static void Main(string[] args)

{

int firstNumber, secondNumber;

// Assigning a constant to variable

firstNumber = 10;

Console.WriteLine("First Number = {0}", firstNumber);

// Assigning a variable to another variable

secondNumber = firstNumber;

Console.WriteLine("Second Number = {0}", secondNumber);

}

}

}

When we run the program, the output will be:

First Number = 10

Second Number = 10

This is a simple example that demonstrates the use of assignment operator.

You might have noticed the use of curly brackets { } in the example. We will discuss about them in *string formatting*. For now, just keep in mind that {0} is replaced by the first variable that follows the string, {1} is replaced by the second variable and so on.

**2. Arithmetic Operators**

Arithmetic operators are used to perform arithmetic operations such as addition, subtraction, multiplication, division, etc.

For example,

int x = 5;

int y = 10;

int z = x + y;// z = 15

| C# Arithmetic Operators | | |
| --- | --- | --- |
| Operator | Operator Name | Example |
| + | Addition Operator | 6 + 3 evaluates to 9 |
| - | Subtraction Operator | 10 - 6 evaluates to 4 |
| \* | Multiplication Operator | 4 \* 2 evaluates to 8 |
| / | Division Operator | 10 / 5 evaluates to 2 |
| % | Modulo Operator (Remainder) | 16 % 3 evaluates to 1 |

**Example 2: Arithmetic Operators**

using System;

namespace Operator

{

class ArithmeticOperator

{

public static void Main(string[] args)

{

double firstNumber = 14.40, secondNumber = 4.60, result;

int num1 = 26, num2 = 4, rem;

// Addition operator

result = firstNumber + secondNumber;

Console.WriteLine("{0} + {1} = {2}", firstNumber, secondNumber, result);

// Subtraction operator

result = firstNumber - secondNumber;

Console.WriteLine("{0} - {1} = {2}", firstNumber, secondNumber, result);

// Multiplication operator

result = firstNumber \* secondNumber;

Console.WriteLine("{0} \* {1} = {2}", firstNumber, secondNumber, result);

// Division operator

result = firstNumber / secondNumber;

Console.WriteLine("{0} / {1} = {2}", firstNumber, secondNumber, result);

// Modulo operator

rem = num1 % num2;

Console.WriteLine("{0} % {1} = {2}", num1, num2, rem);

}

}

}

When we run the program, the output will be:

14.4 + 4.6 = 19

14.4 - 4.6 = 9.8

14.4 \* 4.6 = 66.24

14.4 / 4.6 = 3.1304347826087

26 % 4 = 2

Arithmetic operations are carried out in the above example. Variables can be replaced by constants in the statements.

For example,

result = 4.5 + 2.7 ; // result will hold 7.2

result = firstNumber - 3.2; // result will hold 11.2

**3. Relational Operators**

Relational operators are used to check the relationship between two operands. If the relationship is true the result will be true, otherwise it will result in false.

Relational operators are used in decision making and loops.

| C# Relational Operators | | |
| --- | --- | --- |
| Operator | Operator Name | Example |
| == | Equal to | 6 == 4 evaluates to false |
| > | Greater than | 3 > -1 evaluates to true |
| < | Less than | 5 < 3 evaluates to false |
| >= | Greater than or equal to | 4 >= 4 evaluates to true |
| <= | Less than or equal to | 5 <= 3 evaluates to false |
| != | Not equal to | 10 != 2 evaluates to true |

**Example 3: Relational Operators**

using System;

namespace Operator

{

class RelationalOperator

{

public static void Main(string[] args)

{

bool result;

int firstNumber = 10, secondNumber = 20;

result = (firstNumber==secondNumber);

Console.WriteLine("{0} == {1} returns {2}",firstNumber, secondNumber, result);

result = (firstNumber > secondNumber);

Console.WriteLine("{0} > {1} returns {2}",firstNumber, secondNumber, result);

result = (firstNumber < secondNumber);

Console.WriteLine("{0} < {1} returns {2}",firstNumber, secondNumber, result);

result = (firstNumber >= secondNumber);

Console.WriteLine("{0} >= {1} returns {2}",firstNumber, secondNumber, result);

result = (firstNumber <= secondNumber);

Console.WriteLine("{0} <= {1} returns {2}",firstNumber, secondNumber, result);

result = (firstNumber != secondNumber);

Console.WriteLine("{0} != {1} returns {2}",firstNumber, secondNumber, result);

}

}

}

When we run the program, the output will be:

10 == 20 returns False

10 > 20 returns False

10 < 20 returns True

10 >= 20 returns False

10 <= 20 returns True

10 != 20 returns True

**4. Logical Operators**

Logical operators are used to perform logical operation such as and, or. Logical operators operate on boolean expressions (true and false) and returns boolean values. Logical operators are used in decision making and loops.

Here is how the result is evaluated for logical AND and OR operators.

| C# Logical operators | | | |
| --- | --- | --- | --- |
| Operand 1 | Operand 2 | OR (||) | AND (&&) |
| true | true | true | true |
| true | false | true | false |
| false | true | true | false |
| false | false | false | false |

In simple words, the table can be summarized as:

* If one of the operands is true, the OR operator will evaluate it to true.
* If one of the operands is false, the AND operator will evaluate it to false.

**Example 4: Logical Operators**

using System;

namespace Operator

{

class LogicalOperator

{

public static void Main(string[] args)

{

bool result;

int firstNumber = 10, secondNumber = 20;

// OR operator

result = (firstNumber == secondNumber) || (firstNumber > 5);

Console.WriteLine(result);

// AND operator

result = (firstNumber == secondNumber) && (firstNumber > 5);

Console.WriteLine(result);

}

}

}

When we run the program, the output will be:

True

False

**5. Unary Operators**

Unlike other operators, the unary operators operate on a single operand.

| C# unary operators | | |
| --- | --- | --- |
| Operator | Operator Name | Description |
| + | Unary Plus | Leaves the sign of operand as it is |
| - | Unary Minus | Inverts the sign of operand |
| ++ | Increment | Increment value by 1 |
| -- | Decrement | Decrement value by 1 |
| ! | Logical Negation (Not) | Inverts the value of a boolean |

**Example 5: Unary Operators**

using System;

namespace Operator

{

class UnaryOperator

{

public static void Main(string[] args)

{

int number = 10, result;

bool flag = true;

result = +number;

Console.WriteLine("+number = " + result);

result = -number;

Console.WriteLine("-number = " + result);

result = ++number;

Console.WriteLine("++number = " + result);

result = --number;

Console.WriteLine("--number = " + result);

Console.WriteLine("!flag = " + (!flag));

}

}

}

When we run the program, the output will be:

+number = 10

-number = -10

++number = 11

--number = 10

!flag = False

The increment (++) and decrement (--) operators can be used as prefix and postfix. If used as prefix, the change in value of variable is seen on the same line and if used as postfix, the change in value of variable is seen on the next line. This will be clear by the example below.

**Example 6: Post and Pre Increment operators in C#**

using System;

namespace Operator

{

class UnaryOperator

{

public static void Main(string[] args)

{

int number = 10;

Console.WriteLine((number++));

Console.WriteLine((number));

Console.WriteLine((++number));

Console.WriteLine((number));

}

}

}

When we run the program, the output will be:

10

11

12

12

We can see the effect of using ++ as prefix and postfix. When ++ is used after the operand, the value is first evaluated and then it is incremented by 1.

Hence the statement

Console.WriteLine((number++));

prints 10 instead of 11. After the value is printed, the value of number is incremented by 1.

The process is opposite when ++ is used as prefix. The value is incremented before printing. Hence the statement

Console.WriteLine((++number));

prints 12.

The case is the same for the decrement operator (--).

**6. Ternary Operator**

The ternary operator ? : operates on three operands.

It is a shorthand for if-then-else statement. Ternary operator can be used as follows:

variable = Condition? Expression1 : Expression2;

The ternary operator works as follows:

* If the expression stated by Condition is true, the result of Expression1 is assigned to variable.
* If it is false, the result of Expression2 is assigned to variable.

**Example 7: Ternary Operator**

using System;

namespace Operator

{

class TernaryOperator

{

public static void Main(string[] args)

{

int number = 10;

string result;

result = (number % 2 == 0)? "Even Number" : "Odd Number";

Console.WriteLine("{0} is {1}", number, result);

}

}

}

When we run the program, the output will be:

10 is Even Number

To learn more, visit *C# ternary operator*.

**7. Bitwise and Bit Shift Operators**

Bitwise and bit shift operators are used to perform bit manipulation operations.

| C# Bitwise and Bit Shift operators | |
| --- | --- |
| Operator | Operator Name |
| ~ | Bitwise Complement |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise Exclusive OR |
| << | Bitwise Left Shift |
| >> | Bitwise Right Shift |

**Example 8: Bitwise and Bit Shift Operator**

using System;

namespace Operator

{

class BitOperator

{

public static void Main(string[] args)

{

int firstNumber = 10;

int secondNumber = 20;

int result;

result = ~firstNumber;

Console.WriteLine("~{0} = {1}", firstNumber, result);

result = firstNumber & secondNumber;

Console.WriteLine("{0} & {1} = {2}", firstNumber,secondNumber, result);

result = firstNumber | secondNumber;

Console.WriteLine("{0} | {1} = {2}", firstNumber,secondNumber, result);

result = firstNumber ^ secondNumber;

Console.WriteLine("{0} ^ {1} = {2}", firstNumber,secondNumber, result);

result = firstNumber << 2;

Console.WriteLine("{0} << 2 = {1}", firstNumber, result);

result = firstNumber >> 2;

Console.WriteLine("{0} >> 2 = {1}", firstNumber, result);

}

}

}

When we run the program, the output will be:

~10 = -11

10 & 20 = 0

10 | 20 = 30

10 ^ 20 = 30

10 << 2 = 40

10 >> 2 = 2

To learn more, visit *C# Bitwise and Bit Shift operator*.

**8. Compound Assignment Operators**

| C# Compound Assignment Operators | | | |
| --- | --- | --- | --- |
| Operator | Operator Name | Example | Equivalent To |
| += | Addition Assignment | x += 5 | x = x + 5 |
| -= | Subtraction Assignment | x -= 5 | x = x - 5 |
| \*= | Multiplication Assignment | x \*= 5 | x = x \* 5 |
| /= | Division Assignment | x /= 5 | x = x / 5 |
| %= | Modulo Assignment | x %= 5 | x = x % 5 |
| &= | Bitwise AND Assignment | x &= 5 | x = x & 5 |
| |= | Bitwise OR Assignment | x |= 5 | x = x | 5 |
| ^= | Bitwise XOR Assignment | x ^= 5 | x = x ^ 5 |
| <<= | Left Shift Assignment | x <<= 5 | x = x << 5 |
| >>= | Right Shift Assignment | x >>= 5 | x = x >> 5 |
| => | Lambda Operator | x => x\*x | Returns x\*x |

**Example 9: Compound Assignment Operator**

using System;

namespace Operator

{

class BitOperator

{

public static void Main(string[] args)

{

int number = 10;

number += 5;

Console.WriteLine(number);

number -= 3;

Console.WriteLine(number);

number \*= 2;

Console.WriteLine(number);

number /= 3;

Console.WriteLine(number);

number %= 3;

Console.WriteLine(number);

number &= 10;

Console.WriteLine(number);

number |= 14;

Console.WriteLine(number);

number ^= 12;

Console.WriteLine(number);

number <<= 2;

Console.WriteLine(number);

number >>= 3;

Console.WriteLine(number);

}

}

}

When we run the program, the output will be:

15

12

24

8

2

2

14

2

8

1

We will discuss about *Lambda operators* later in tutorial.