**C# Operator Precedence and Associativity**

In this tutorial, we will learn about operator precedence and associativity in C#. This will give us an idea of how an expression is evaluated by the C# compiler.

**C# Operator Precedence**

Operator precedence is a set of rules which defines how an expression is evaluated. In C#, each [C# operator](https://www.programiz.com/csharp-programming/operators) has an assigned priority and based on these priorities, the expression is evaluated.

**For example**, the precedence of multiplication (\*) operator is higher than the precedence of addition (+) operator. Therefore, operation involving multiplication is carried out before addition.

Take a look at the statement below.

int x = 4 + 3 \* 5;

What will be the value of x after executing this statement?

The operand 3 is associated with + and \*. As stated earlier, multiplication has a higher precedence than addition. So, the operation 3 \* 5 is carried out instead of 4 + 3. The value of variable x will be 19.

If addition would have a higher precedence, 4 + 3 would be evaluated first and the value of x would be 35.

**Operator Precedence Table**

The higher the precedence of operator is, the higher it appears in the table

| C# Operator Precedence | |
| --- | --- |
| Category | Operators |
| Postfix Increment and Decrement | ++, -- |
| Prefix Increment, Decrement and Unary | ++, --, +, -, !, ~ |
| Multiplicative | \*, /, % |
| Additive | +, - |
| Shift | <<, >> |
| Relational | <, <=, >, >= |
| Equality | ==, != |
| Bitwise AND | & |
| Bitwise XOR | ^ |
| Bitwise OR | | |
| Logical AND | && |
| Logical OR | || |
| Ternary | ? : |
| Assignment | =, +=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>= |

The assignment operators have the lowest precedence while the postfix increment and decrement operators have the highest precedence.

**Example 1: Operator Precedence**

using System;

namespace Operator

{

class OperatorPrecedence

{

public static void Main(string[] args)

{

int result1;

int a = 5, b = 6, c = 4;

result1 = --a \* b - ++c;

Console.WriteLine(result1);

bool result2;

result2 = b >= c + a;

Console.WriteLine(result2);

}

}

}

When we run the program, the output will be:

19

False

Let's understand how the expression is evaluated in the program.

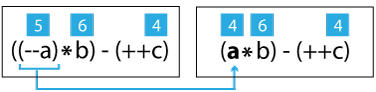
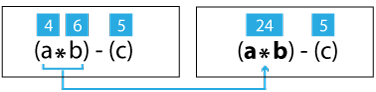
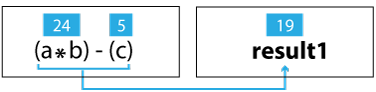
The precedence of -- and ++ is higher than \*, and precedence of \* is higher than -. Hence the statement,

result1 = --a \* b - ++c;

is equivalent to

result1 = ((--a)\*b)-(++c);

The expression inside parentheses is always evaluated first no matter what the precedence of operators outside it is.

* At first, (--a) is evaluated resulting into 4.  
  
* Then (++c) is evaluated resulting into 5.  
  
* Now, (a \* b) is evaluated resulting into 24.  
  
* Finally, the subtraction is carried out resulting into 19.  
  
* Hence the final value of result1 will be 19.

In the next expression, the precedence of + is higher than >=. So, c and a is added first and the sum is compared with b to produce false.

**Associativity of Operators in C#**

In the previous section, we discussed about operator precedence. If two operators with different precedence are used, the operator with higher precedence is evaluated first.

But what if both the operators have same precedence?

In such case, the expression is evaluated based on the associativity of operator (left to right or right to left).

**For example:**

int a = 5, b = 6, c = 3;

int result = a \* b / c;

Here, both \* and / have the same precedence. But since the associativity of these operators is from **left to right**, a \* b is evaluated first and then division is carried out. The final result of this expression will be 10.

In this particular example, the associativity does not really matter. Because even if division was carried out before multiplication, the result would be unaffected.

Let's take a look at another example.

int a = 5, b = 6, c = 3;

a = b = c;

The associativity of = operator is from **right to left**. So, the value of c (i.e. 3) is assigned to b, and then the value of b is assigned to a. So, after executing this statement, the values of a, b and c will be 3.

The table below shows the associativity of C# operators:

| C# Associativity of operators | | |
| --- | --- | --- |
| Category | Operators | Associativity |
| Postfix Increment and Decrement | ++, -- | Left to Right |
| Prefix Increment, Decrement and 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 |
| Ternary | ? : | Right to Left |
| Assignment | =, +=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>= | Right to Left |

Almost all the operators have associativity from left to right. The operators having associativity from right to left are:

* Unary operators
* Prefix Increment and Decrement Operators
* Ternary Operator
* Assignment Operators

**Example 2: Associativity of Operators**

using System;

namespace Operator

{

class OperatorPrecedence

{

public static void Main(string[] args)

{

int a = 5, b = 6, c = 3;

int result = a \* b / c;

Console.WriteLine(result);

a = b = c;

Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);

}

}

}

When we run the program, the output will be:

10

a = 3, b = 3, c = 3