Zakho Technical Institute

Department of Information Technology

Object Oriented Programming

1. Conditional Statements In C#

Lecturers:

Sipan M. Hameed

www.sipan.dev

2024-2025

Table of Contents

C# Operators

Operators in C# are some special symbols that perform some action on operands. In mathematics, the plus symbol (+) do the sum of the left and right numbers. In the same way, C# includes various operators for different types of operations.

The following example demonstrates the + operator in C#.

Example: + Operator

```
int x = 5 + 5;
int y = 10 + x;
int z = x + y;
```

In the above example, + operator adds two number literals and assign the result to a variable. It also adds the values of two int variables and assigns the result to a variable.

Some operators behave differently based on the type of the operands. For example, + operator concatenates two strings.

Example: + Operator with Strings

```
string greet1 = "Hello " + "World!";
string greet2 = greeting + name;
```

Note:

There are two types of operators in C#, Unary operators and Binary operators. Unary operators act on single operand, whereas binary operators act on two operands (left-hand side and right-hand side operand of an operator).

C# includes the following categories of operators:

Arithmetic operators
Assignment operators
Comparison operators

Equality operators

Boolean logical operators

Betwise and shift operators

Member access operators

Type-cast operators

Pointer related operators

Arithmetic Operators

The arithmetic operators perform arithmetic operations on all the numeric type operands such as sbyte, byte, short, ushort, int, uint, long, ulong, float, double, and decimal.

Operator	Name	Description	
+	Addition	Computes the sum of left and right operands.	int x = 5 + 5;
-	Subtraction	Subtract the right operand from the left operand	int x = 5 - 1;
*	Multiplication	Multiply left and right operand	int x = 5 * 1;
/	Division	Divides the left operand by the right operand	int x = 10 / 2;
%	Reminder	Computes the remainder after dividing its left operand by its right operand	int x = 5 % 2;
++	Unary increment	Unary increment ++ operator increases its operand by 1	x++
--	Unary decrement	Unary decrement -- operator decreases its operand by 1	x--
+	Unary plus	Returns the value of operand	+5
-	Unary minus	Computes the numeric negation of its operand.	-5

Assignment Operators

The assignment operator = assigns its right had value to its left-hand variable, property, or indexer. It can also be used with other arithmetic, Boolean logical, and bitwise operators.

Operator	Name	Description	
=	Assignment	Assigns its right had value to its left-hand variable, property or indexer.	
x op= y	Compound assignment	Short form of x =x op y where op = any arithmetic, Boolean logical, and bitwise operator.	x += 5;
??=	Null-coalescing assignment	C# 8 onwards, ??= assigns value of the right operand only if the left operand is null	x ??= 5;

Comparison Operators

Comparison operators compre two numeric operands and returns true or false.

Operator	Description	Example
<	Returns true if the right operand is less than the left operand	x < y;
>	Returns true if the right operand is greater than the left operand	x > y;
<=	Returns true if the right operand is less than or equal to the left operand	x <= y
>=	Returns true if the right operand is greater than or equal to the left operand	x >= y;

Equality Operators

The equality operator checks whether the two operands are equal or not.

Operator	Description	
==	Returns true if operands are equal otherwise false.	x == y;
!=	Returns true if operands are not equal otherwise false.	x != y;

Boolean Logical Operators

The Boolean logical operators perform a logical operation on bool operands.

Operator	Description	Example
!	Reverses the bool result of bool expression. Returns false if result is true and returns true if result is false.	!false
&&	Computes the logical AND of its bool operands. Returns true both operands are true, otherwise returns false.	x && y;
||	Computes the logical OR of its bool operands. Returns true when any one operand is true.	x || y;

Operator Evaluation & Precedence

Evaluation of the operands in an expression starts from left to right. If multiple operators are used in an expression, then the operators with higher priority are evaluated before the operators with lower priority.

The following table lists operators starting with the higher precedence operators to lower precedence operators.

Operators	Category
x.y, x?.y, x?[y], f(x), a[i], x++, x--, new, typeof, checked, unchecked, default, nameof, delegate, sizeof, stackalloc, x->y	Primary
+x, -x, !x, ~x, ++x, --x, ^x, (T)x, await, &x, *x, true and false	Unary
x..y	Range
x * y, x / y, x % y	Multiplicative
x + y, x – y	Additive
x << y, x >> y	Shift
x < y, x > y, x <= y, x >= y, is, as	Relational and type-testing
x == y, x != y	Equality
x & y	Boolean logical AND
x ^ y	Boolean logical XOR
x | y	Boolean logical OR
x && y	Conditional AND
x || y	Conditional OR
x ?? y	Null-coalescing operator
c ? t : f	Conditional operator
x = y, x += y, x -= y, x *= y, x /= y, x %= y, x &= y, x |= y, x ^= y, x <<= y, x >>= y, x ??= y, =>	Assignment and lambda declaration

The following example demonstrates operator precedence:

Example: Operator Precedence

```
int a = 5 + 3 * 3;
int b = 5 + 3 * 3 /  2;
int c = (5 + 3) * 3 /  2;
int d = (3 * 3) * (3 / 3 + 5);
```

if, else if, else Statements

C# provides many decision-making statements that help the flow of the C# program based on certain logical conditions. Here, you will learn about if, else if, else, and nested if else statements to control the flow based on the conditions.

C# includes the following flavors of if statements:

1. if statement
2. else-if statement
3. else statement

C# if Statement

The if statement contains a boolean condition followed by a single or multi-line code block to be executed. At runtime, if a boolean condition evaluates to true, then the code block will be executed, otherwise not.

Syntax:

```
if(condition)
{
    // code block to be executed when if condition evaluates to true
}
```

Example: if Statement

```
int i = 10, j = 20;

if (i < j)
{
    Console.WriteLine("i is less than j");
}

if (i > j)
{
    Console.WriteLine("i is greater than j");
}
```

Output:

```
i is less than j
```

In the above example, a boolean condition in the first if statement i < j evaluates to true, so the C# compiler will execute the following code block. The second if statement's condition i > j evaluates to false, so the compiler will not execute its code block.

The conditional expression must return a boolean value, otherwise C# compiler will give a compile-time error.

Example: Wrong if Statement

int i = 10, j = 20;

```

if (i + 1)
{
    Console.WriteLine("i is less than j");
}

if (i + j)
{
    Console.WriteLine("i is greater than j");
}
```

You can call a function in the if statement that returns a boolean value.

Example: Calling Function as Condition

```
static void Main(string[] args)
{
    int i = 10, j = 20;

    if (isGreater(i, j))
    {
        Console.WriteLine("i is less than j");
    }

    if (isGreater(j, i))
    {
        Console.WriteLine("j is greater than i");
    }
}

static bool isGreater(int i, int j)
{
    return i > j;
}
```

else if Statement

Multiple else if statements can be used after an if statement. It will only be executed when the if condition evaluates to false. So, either if or one of the else if statements can be executed, but not both.

```
Syntax:
if(condition1)
{
    // code block to be executed when if condition1 evaluates to true
}
else if(condition2)
{
    // code block to be executed when
    //      condition1 evaluates to flase
    //      condition2 evaluates to true
}
else if(condition3)
{
    // code block to be executed when
    //      condition1 evaluates to flase
    //      condition2 evaluates to false
    //      condition3 evaluates to true
}
```

The following example demonstrates else if statements.

Example: else if Statements

```
int i = 10, j = 20;


if (i == j)
{
    Console.WriteLine("i is equal to j");
}
else if (i > j)
{
    Console.WriteLine("i is greater than j");
}
else if (i < j)
{
    Console.WriteLine("i is less than j");
}
```

Output:

```
i is less than j
```

else Statement

The else statement can come only after if or else if statement and can be used only once in the if-else statements. The else statement cannot contain any condition and will be executed when all the previous if and else if conditions evaluate to false.

Example: else Statement

```
int i = 20, j = 20;


if (i > j)
{
    Console.WriteLine("i is greater than j");
}
else if (i < j)
{
    Console.WriteLine("i is less than j");
}
else
{
    Console.WriteLine("i is equal to j");
}
```

Output:

```
i is equal to j
```

Nested if Statements

C# supports if else statements inside another if else statements. This are called nested if else statements. The nested if statements make the code more readable.

Syntax:

```
if(condition1)
{
   if(condition2)
    {
        // code block to be executed when
        //      condition1 and condition2 evaluates to true
    }
    else if(condition3)
    {
        if(condition4)
        {
            // code block to be executed when
            //      only condition1, condition3, and condition4 evaluates to true
        }
        else if(condition5)
        {
            // code block to be executed when
            //      only condition1, condition3, and condition5 evaluates to true
        }
        else
        {
            // code block to be executed when
            //      condition1, and condition3 evaluates to true
            //      condition4 and condition5 evaluates to false
        }
    }
}
```

The following example demonstrates the nested if else statements.

Example: Nested if else statements

```
int i = 10, j = 20;


if (i != j)
{
    if (i < j)
    {
        Console.WriteLine("i is less than j");
    }
    else if (i > j)
    {
       Console.WriteLine("i is greater than j");
    }
}
else
    Console.WriteLine("i is equal to j");
```

Output:

```
i is less than j
```

Ternary Operator ?:

C# includes a decision-making operator ?: which is called the conditional operator or ternary operator. It is the short form of the if else conditions.

Syntax:

```
condition ? statement 1 : statement 2
```

The ternary operator starts with a boolean condition. If this condition evaluates to true then it will execute the first statement after ?, otherwise the second statement after : will be executed.

The following example demonstrates the ternary operator.

Example: Ternary operator

```
int x = 20, y = 10;

var result = x > y ? "x is greater than y" : "x is less than y";

Console.WriteLine(result);
```

output:

```
x is greater than y
```

Above, a conditional expression x > y returns true, so the first statement after ? will be execute.

The following executes the second statement.

Example: Ternary operator

```
int x = 10, y = 100;

var result = x > y ? "x is greater than y" : "x is less than y";

Console.WriteLine(result);
```

output:

```
x is less than y
```

Thus, a ternary operator is short form of if else statement. The above example can be re-write using if else condition, as shown below.

Example: Ternary operator replaces if statement

```
int x = 10, y = 100;


if (x > y)
    Console.WriteLine("x is greater than y");
else
    Console.WriteLine("x is less than y");
```

output:

x is greater than y

```
Nested Ternary Operator
```

Nested ternary operators are possible by including a conditional expression as a second statement.

Example: Nested ?:

```
int x = 10, y = 100;


string result = x > y ? "x is greater than y" :
                    x < y ? "x is less than y" :
                        x == y ? "x is equal to y" : "No result";


Console.WriteLine(result);
```

The ternary operator is right-associative. The expression a ? b : c ? d : e is evaluated as a ? b : (c ? d : e), not as (a ? b : c) ? d : e.

Example: Nested ?:

```
var x = 2, y = 10;


var result = x * 3 > y ? x : y > z? y : z;
Console.WriteLine(result);
```

Switch Case statement

Switch statement is the short and better way to execute the If-Else-If statement. If we check the value multiple times, then the process goes to complex and performance would be slow. In that case we use switch case statement which acts as a substitute for long If-Else-If ladder that is used to test a series of conditions. A switch statement contains one or more case labels which are tested against the switch expression. If value is matched with particular case then only that case will be executed and rest of all will be untouched.

Let's take an example to understand:

```
public class MyClass
{
    static void Main(string[] args)
    {
        char grade = 'B';

        switch (grade)
        {
            case 'A':
                Console.WriteLine("Excellent!");
                break;
            case 'B':
            case 'C':
                Console.WriteLine("Well done");
                break;
            case 'D':
                Console.WriteLine("You passed");
                break;
            case 'F':
                Console.WriteLine("Better try again");
                break;
            default:
                Console.WriteLine("You Failed!");
                break;
        }
    }
}
```