What are Statements, Expression and Operators in C#?

A C# program is a set of tasks that perform to achieve the overall functionality of the program.

To perform the tasks, programmers provide instructions. These instructions are called statements.

Statements are referred to as logical grouping of variables, operators, and C# keywords that perform a special task.

A C# program contains multiple statements grouped in blocks. A block is a code segment enclosed in curly braces. {}

Statements in C#

Statements are used to specify the input, the process, and the output tasks of a program. Statements can consist of -:

- **Data types**
- **Variables**
- **Operators**
- **Constants**
- **Literals**
- **Keywords**
- **Escape Sequence Characters**

Statements help you build a logical flow in the program. With the help of statements, you can -:

- **Initialize variables and objects**
- **Take the input**
- **Call a method of a class**
- **Display the output**

Similar to statements in C and C++, the C# statements are classified into several categories:

- **Selections Statements**
- **Iterations Statements**
- **Jump Statements**
- **Exception Handling Statements**

Expression in C#

Expressions are used to manipulate data. Like in mathematics, expression in programming languages,

Including C# are constructed from the operands and operators.

Example -: 2 + 2 (*Here 2 is an operand and + is a operators)*

An expression statements in C# ends with a semicolon (;).

The Flowing code demonstrates an example for expression:

```
simple Interest   = principal * time * rate / 100;
```

Operators in C#

Expressions in C# comprise one or more operators that performs some operations on variables.

An operation is actions perform on single or multiple values stored in variables in order to modify them or to generate a new value with the help of minimum one symbol and a value.

The symbol is called an operator and it determine the type of action to be performed on the value.

An operand might be a complex expression.

For example, (X * Y) + (Y – X) is a complex expression, Where the + operator is used to join two operands.

The value on which the operation is to be performed is called an operand.

Operators are used to simplify expressions in C#; there is predefined set of operators used to perform various types of operations.

Classification of Operators

Operators are classified in 3 categories:

1. **UNARY OPERATORS**

- Unary Operators are those operators that operate on a single operand to perform an operation like increment, decrement, negation, or logical NOT.
  They change or return a new value based on only one variable or constant.

2. **BINARY OPERATORS**

- Binary Operators are operators that act on two operands (values or variables) to perform a specific operation such as addition, subtraction, multiplication, comparison, or logical operation.

- ➡ **In simple words:**
  Binary operators need two values to work, e.g., a + b, x > y, num1 && num2.

3. **TERNARY OPERATORS**

- Ternary Operator is an operator that works on three operands.
  It is also known as the Conditional Operator and is used to make decisions in a single line.

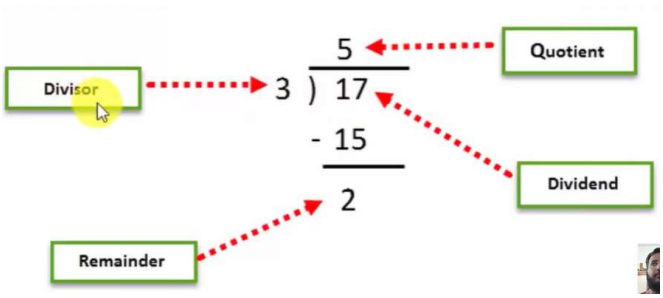
These are classified into seven categories based on the action the perform on values:

- **Arithmetic Operators**
- **Relational Operators**
- **Logical Operators**
- **Bitwise Operators**
- **Conditional Operators**
- **Increment and Decrement Operators**
- **Assignment Operators**

Arithmetic Operators in C#

Arithmetic Operators are binary operators because they work with two operands, with the operator being placed in between the operands.

- + (addition)
- -(subtraction)
- (multiplication)
- / (division)
- % (modulus)



Relational Operators in C#

Relational Operators make a comparison between two operands and return a Boolean value, true or false.

- == (double equal to)
- ! = (not equal to)
- > (greater than)
- < (less than)
- >= (greater than equal to)
- <= (less than equal to)

Logical OR Conditional Operators in C#

Logical Operators perform Boolean logical operations on both the operands. They return a Boolean value based on the logical operator used.

There are two types of conditional operators.

Conditional AND (&&)

OPERATOR	CONDITION 1	CONDITION 2	RESULT
AND &&	TRUE	TRUE	TRUE
	TRUE	FALSE	FALSE
	FALSE	TRUE	FALSE
	FALSE	FALSE	FALSE

Conditional OR (||).

OPERATOR	CONDITION 1	CONDITION 2	RESULT
OR ||	TRUE	TRUE	TRUE
	TRUE	FALSE	TRUE
	FALSE	TRUE	TRUE
	FALSE	FALSE	FALSE

Assignment Operators in C#

Assignment Operator are used to assign the value of the right-side operand to the operand on the left side using the equal to operator (=).

Int a = 20;

The assignment operators are divided into two categories in C#. These are as follows.

Simple assignment operators. The simple assignment operator is =,

which is used to assign a value or result of an expression to a variable.

Compound assignment operators: The compound assignment operators are formed by combining the simple assignment operator with the arithmetic operators.

Compound assignment Operators:

- +=
- -=
- *=
- /=
- %=

Increment & Decrement Operators in C#

Two of the most common calculations performed in programming are increasing and decreasing the value of the variable by 1.

In C#, the increment operator (+ +) is used to increase the value by 1

While the decrement operator (- -) is used to decrease the value by 1.

If the operator is placed before the operand, the expression is called pre-increment or pre-decrement.

If the operator is placed after the operand, the expression is called post-increment or post-decrement.

Ternary Operators in C#

C# includes a special type of decision-making operator? : called the ternary operator.

Syntex:

Boolean Expression? First Statement: Second Statement

Selection Construct or Statement

Selection Construct is a programming construct supported by C# that controls the flow of a program.

Executes a particular block of statements based on a Boolean condition, which is an expression returning true or false.

Is referred to as a decision-making construct.

Allow you to take logical decisions about executing different blocks of a program to achieve the required logical output.

C# supports the following decision-making constructs:

- **If…else**
- **If…else…if**
- **Nested if**
- **Switch…case**
- **Nested switch case**

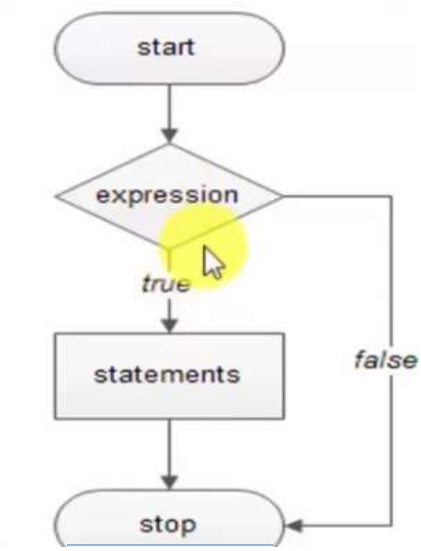
The if statement

The if statement allows you to execute a block of statements after evaluating the specified logical condition.

The if statement starts with the if keyword and is followed by the condition.

If the condition evaluates to true, the block of statements following the if statement is executed.

If the condition evaluates to false, the block of statements following the if statement after the block is executed.



The if-else-if statement

The if…else...if construct allows you to check multiple condition to execute a different block of code for each condition.

It is also referred to as if else if ladder.

The construct starts with the if statement followed by multiple else if statements followed by an optional else block.

The condition specified in the if…else…if construct is evaluated sequentially.

The execution states from the if statement. If a condition evaluates to false, the conditional specified in the following else…if statement is evaluated.

Nested if

The nested if conditions consist of multiple if statements.

The nested if construct starts with the if statement, which is called the outer if statement, and contains multiple if statements, which are called inner if statements

Switch Case in C#

A program is different to comprehend when there are too many if statements representing multiple selection constructs.

The avoid multiple if statements, in certain cases, the switch… case approach can be used as an alternative.

The Switch…case statement is used when a variable needs to be compared against different values.

In C#, the flow of execution from one case statement is not allowed to continue to the next case statement and is referred to as the 'no fall through' rule of C#

Nested Switch Case

C# allows the switch… case construct to be nested. That is, a case block of a switch case construct can contain another switch case construct.

Also, the case constants of the inner switch…case construct can have values that are identical to the case constants of the outer construct.

Loops Constructs

Loops allow you to executed a single statement or a block of statement repetitively.

The most common uses of loops include displaying a series of numbers and taking repetitive input.

In software programming, a loop construct contains a condition that help the compiler identify the number of times a specific block will be executed.

If the condition is not specified, the loop continues infinitely and is termed as an infinite loop.

The loop constructs are also referred to as iteration statements.

C# Supports four types of loop constructs such as:

- **The for loop**
- **The while loop**
- **The do…while loop**
- **The foreach loop**

The For loop in C#

The for statements is similar to the while statement in its function.

The statements within the body of the loop are executed as long the conditional is true.

There are 3 things in For Loop:

- **Initialization**
- **Condition**
- **Increment/Decrement**

Here too, the condition is checked before the statements are executed.

While Loop in C#

The while loop is used to execute a block of code repetitively as long as the condition of the loop remains true.

The while loop consists of the while statement, which begins with the while keyword followed by a Boolean condition.

The while loop consists of the while statement, which begins with the while keyword followed by a Boolean condition.

If the condition evaluates to true, block of statement after the while statement is executed.

After each iteration, the control is transferred back to the while statement and the condition is checked again for another round of execution.

When the condition is evaluated to false, the block of statements following the while statements is ignored and the statements appearing after the bock is executed by the compiler.

Do-While Loop in C#

The do-while loop is similar to the while loop; however, it is always executed at least once without the condition being checked.

The loop starts with the do keyword and is followed by a block of executed statements.

The while statement along with the condition appears at the end of this block.

The statements in the do-while loop are executed as long as the specified condition remains true.

When the condition evaluates to false, the block of statements after the do keyword are ignored and the immediate statement after the while statement is executed.

Nested For Loop

The nested for loop consists of multiple for statements.

When one for loop is enclosed inside another for loop, the loops are said to be nested.

The for loop that enclose the other for loop is referred to as the outer for loop whereas the enclosed for loop is referred to as the inner for loop.

The outer for loop determines the number of times the inner for loop while be invoked.

Jump statements in C#

Jump statements are used to transfer control from one point in a program to another.

C# Supports four types of jump statements.

- **Break**
- **Continue**
- **Goto**
- **Return**

Jump statements are used to transfer control form one point in a program to another.

There will be situations where you need to exit out of a loop prematurely and continue with program.

In such cased, jump statements are used. Jump statement unconditionally transfer control of a program to a different location.

The location to which a jump statement transfers control is called the target of the jump statement.

Break statement

The break statement is used in the selection and loop constructs.

It is most widely used in the switch…case construct and in the for and while loops.

The break statement is denoted by the break keywork. In the switch…case construct, it is used to terminate the execution of the construct.

In this case, the control passes to the next statement following the loop.

Continue Statement

The continue statement is used to end the current iteration of the loop and transfer the program control back to the beginning of the loop.

The statements of the loop following the continue statement are ignored in the current iteration.

Goto Statement

The goto statement allows you to directly executed a labeled statement or a labeled block of statements.

A labeled block or a labeled statement starts with a label. A label is an identifier ending with a colon.

A single labeled block can be referred by more than one goto statements.

The goto statement is denoted by the goto keyword