C# is a programming language of .Net Framework.

What is C#

C# is pronounced as "C-Sharp". It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.

By the help of C# programming language, we can develop different types of secured and robust applications:

- Window applications
- Web applications
- Distributed applications
- Web service applications
- Database applications etc.

C# is approved as a standard by ECMA and ISO. C# is designed for CLI (Common Language Infrastructure). CLI is a specification that describes executable code and runtime environment.

C# programming language is influenced by C++, Java, Eiffel, Modula-3, Pascal etc. languages.

Different types of applications that can be developed using C# are:

- Web applications.
- Window applications.
- Other Desktop software.
- Distributed applications.
- Database programs.
- Hardware-level programming.
- Virus and Malware.
- GUI based applications.

Features of C#

There are various features of C# which makes the programming language different and widely used. These are:

- Learning C# is very easy.
- It is a general purpose, easy integrating programming language.
- It is a highly structured programming language.
- Object-oriented concepts can be implemented efficiently using this language.

- It is platform independent, which means the programs written in C# can be executed in a variety of computing environment.
- Efficient programming can be done using this language.
- It is a high-level programming language.
- GUI applications can be developed very easily using this language.

Programming Features of C#

Other reasons why programming in C# language is so strong and popular are:

- It deals with automatic garbage collections.
- It has a lot of standard libraries.
- Some properties and events can make programming smarter.
- Delegates and concepts of event management can also help in making the language strong.
- It supports Multi-threading.
- It has the concepts of Indexers.
- Generic concepts are easy to use in C#.
- It provides LINQ and Lambda expressions.

Java vs C#

There are many differences and similarities between Java and C#. A list of top differences between Java and C# are given below:

No.	Java	C#
1)	Java is a **high level, robust, secured and object-oriented programming** language developed by Oracle.	C# is an **object-oriented programming** language developed by Microsoft that runs on .Net Framework.
2)	Java programming language is designed to be run on a Java platform, by the help of **Java Runtime Environment (JRE).**	C# programming language is designed to be run on the **Common Language Runtime (CLR).**
3)	Java type safety is safe.	C# type safety is unsafe.

4)	In java, built-in data types that are passed by value are called **primitive types.**	In C#, built-in data types that are passed by value are called **simple types.**
5)	Arrays in Java are direct specialization of **Object.**	Arrays in C# are specialization of **System.**
6)	Java does not support **conditional compilation.**	C# supports conditional compilation using preprocessor directives.
7)	Java doesn't support goto statement.	C# supports goto statement.
8)	Java doesn't support **structures and unions.**	C# supports structures and unions.
9)	Java supports checked exception and unchecked exception.	C# supports unchecked exception.

C# History

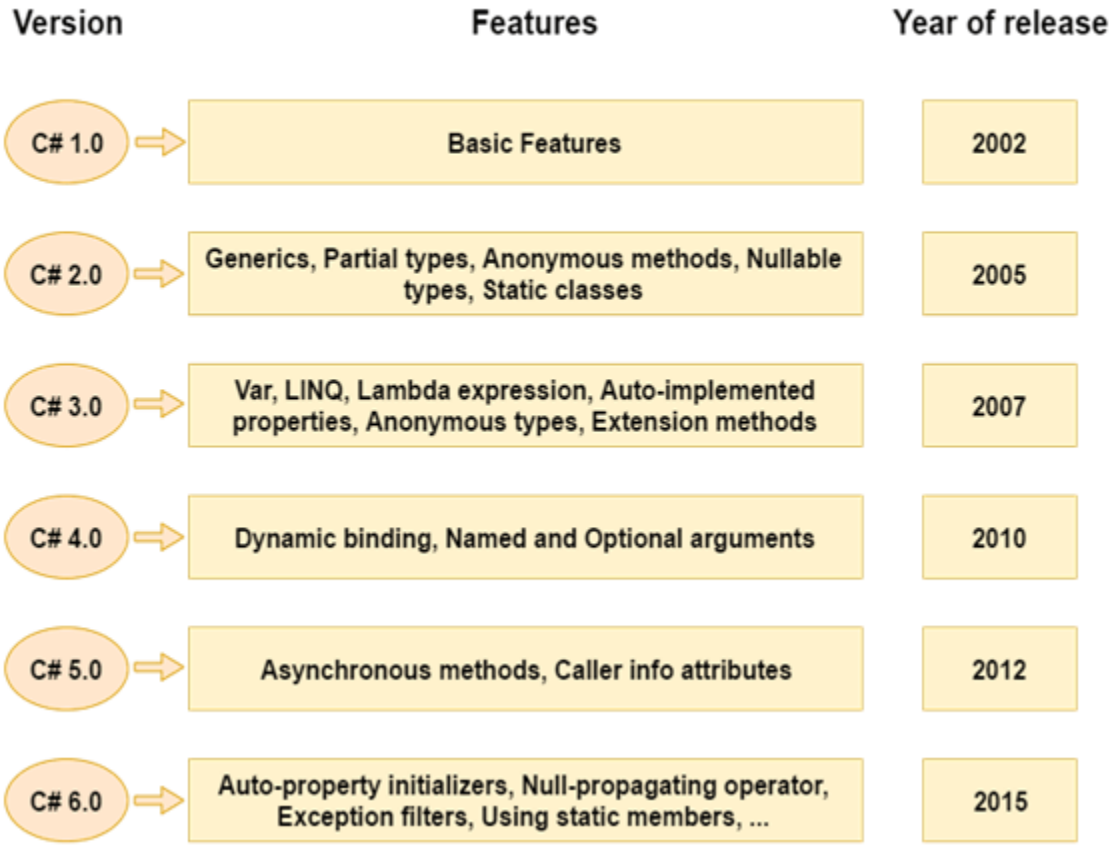
C# is pronounced as **"C-Sharp"**. It is an object-oriented programming language provided by **Microsoft** that runs on **.Net Framework.**

Anders Hejlsberg is known as the **founder of C# language.**

It is based on **C++ and Java**, but it has many additional extensions used to perform component oriented programming approach.

C# has evolved much since their first release in the year **2002**. It was introduced with **.NET Framework 1.0** and the current version of C# is 7.0. the important features introduced in each version of C# are given below.

C# Version History



C# Example: Hello World

In C# programming language, a simple "hello world" program can be written by multiple ways. Let's see the top 4 ways to create a simple C# example:

- Simple Example
- Using System
- Using public modifier
- Using namespace

C# Simple Example

```
class Program
   {
      static void Main(string[] args)
      {
         System.Console.WriteLine("Hello World!");
      }
   }
```

Output:

```
Hello World!
```

> Description
>
> **class:** is a keyword which is used to define class.
>
> **Program:** is the class name. A class is a blueprint or template from which objects are created. It can have data members and methods. Here, it has only Main method.
>
> **static:** is a keyword which means object is not required to access static members. So it saves memory.
>
> **void:** is the return type of the method. It does't return any value. In such case, return statement is not required.
>
> **Main:** is the method name. It is the entry point for any C# program. Whenever we run the C# program, Main() method is invoked first before any other method. It represents start up of the program.
>
> **string[] args:** is used for command line arguments in C#. While running the C# program, we can pass values. These values are known as arguments which we can use in the program.

System.Console.WriteLine("Hello World!"): Here, System is the namespace. Console is the class defined in System namespace. The WriteLine() is the static method of Console class which is used to write the text on the console.

C# Example: Using System

If we write *using System* before the class, it means we don't need to specify System namespace for accessing any class of this namespace. Here, we are using Console class without specifying System.Console.

```
1.      using System;
2.         class Program
3.         {
4.            static void Main(string[] args)
5.            {
6.               Console.WriteLine("Hello World!");
7.            }
8.         }
```

Output:

```
Hello World!
```

C# Example: Using public modifier

We can also specify *public* modifier before class and Main() method. Now, it can be accessed from outside the class also.

```
1.      using System;
2.         public class Program
3.         {
4.            public static void Main(string[] args)
5.            {
6.               Console.WriteLine("Hello World!");
7.            }
8.         }
```

Output:

```
Hello World!
```

C# Example: Using namespace

We can create classes inside the namespace. It is used to group related classes. It is used to categorize classes so that it can be easy to maintain.

```
1.      using System;
2.      namespace ConsoleApplication1
3.      {
```

```
4.              public class Program
5.              {
6.                 public static void Main(string[] args)
7.                 {
8.                    Console.WriteLine("Hello World!");
9.                 }
10.             }
11.         }
```

Output:

```
Hello World!
```

Table of Contents

What Are Variables in C#?

Variables are specific names given to locations in the memory for storing and dealing with data. Values of a variable can be changed or reused as many times as possible. Every variable name has a specific type that resolves the size and layout of memory the variable will hold as well as the range of values that variable within your program can hold. Also, programmers can determine what type of operations that variable can be applied for.

Types of C# Variables

The basic types of variables that can be formed in a C# program are:

Type	Description
Integral types	sbyte, byte, short, ushort, int, uint, long, ulong, and char
Floating point types	float and double
Decimal types	decimal
Boolean types	true or false values, as assigned

Nullable types	Nullable data types

C# also permits programmers to define other types of variables and their values like the enum and reference types of variables like classes.

Define Variables in C#

For implementing variables in a C# program, you have to define them before using. To do this the syntax is:

Syntax:

```
<data_type> <variable_names>;
```

Here data_types will be a valid data type (such as char, int, float, double or any other user-defined data type) of C# and the variable_names will be the set of variable names which are valid C# identifiers separated by commas.

Example:

```
char s, chrr;

int a, b, c;

float pi, sal;

double aadharno;
```

Initializing Variables in C#

A C #variable gets initialized using the assignment operator which is the equal sign. We will learn more about different operators in the subsequent chapters. The syntax for initializing a variable in C# is:

Syntax:

```
<data_type> <variable_name> = value;
```

Some common Examples are:

Example:

```
char ch = 'g';


int xy = 6, roll = 42;


byte b = 22;


double pi = 3.14159;


float salary = 20000.0f;
```

Accepting Values in a Program

There is a particular function of the Console class that provides the function *Readline()* to take input from the user for storing them in a variable, which is ultimately a named memory location.

Let us see how to use this function name with any variable to fetch (using the keyboard) and assign values to that variable:

Example:

```
double sal;
sal = Convert.ToDouble(Console.ReadLine());
```

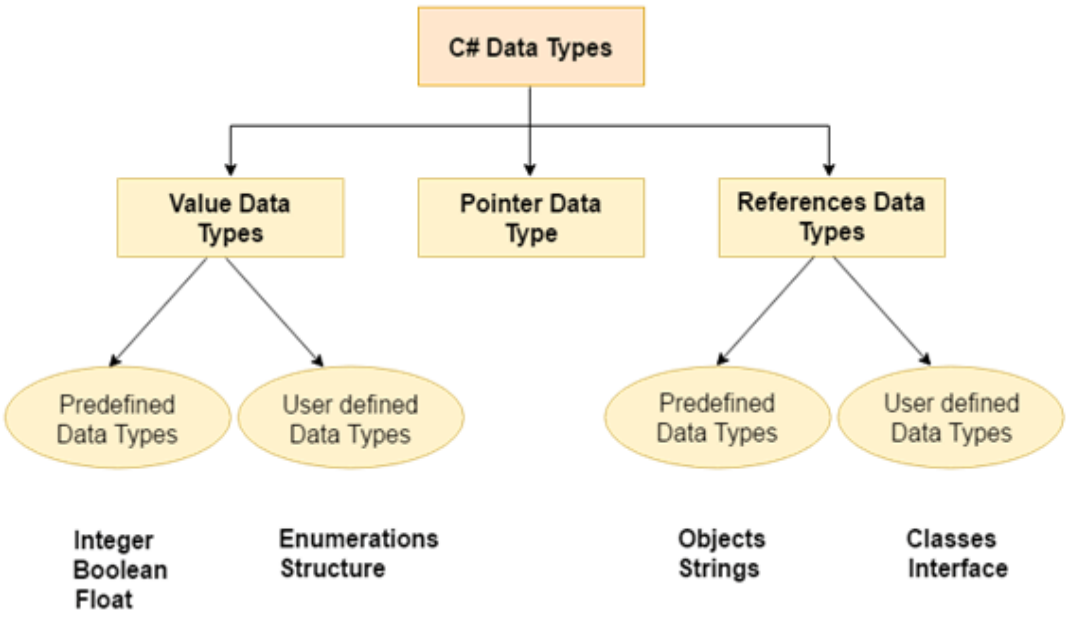
In the above code snippet, the first line will declare a variable as a double type. The second line will first execute the right side which will wait for the user to input any number and will

convert the input values to double using the *Convert.ToDouble()* and will finally assign that value to 'sal' variable using the *assignment operator* (=).

C# Data Types

As you have already known that while declaring a variable in C#, you have to specify the data type with the variable name. This is how you tell the compiler what type of data the variable will hold. Almost all programming language needs the concept of the data type. Since C# is a strongly typed language, it is essential to inform the compiler what kind of data these memory locations will hold.

A data type specifies the type of data that a variable can store such as integer, floating, character etc.



There are 3 types of data types in C# language.

Types	Data Types
Value Data Type	short, int, char, float, double etc
Reference Data Type	String, Class, Object and Interface
Pointer Data Type	Pointers

Value Data Type

The value data types are integer-based and floating-point based. C# language supports both signed and unsigned literals.

There are 2 types of value data type in C# language.

1) Predefined Data Types - such as Integer, Boolean, Float, etc.

2) User defined Data Types - such as Structure, Enumerations, etc.

The memory size of data types may change according to 32 or 64 bit operating system.

Let's see the value data types. It size is given according to 32 bit OS.

Data Types	Memory Size	Range
char	1 byte	-128 to 127
signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 127
short	2 byte	-32,768 to 32,767

signed short	2 byte	-32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	4 byte	-2,147,483,648 to -2,147,483,647
signed int	4 byte	-2,147,483,648 to -2,147,483,647
unsigned int	4 byte	0 to 4,294,967,295
long	8 byte	?9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
signed long	8 byte	?9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long	8 byte	0 - 18,446,744,073,709,551,615
float	4 byte	$1.5 * 10^{-45}$ - $3.4 * 10^{38}$, 7-digit precision
double	8 byte	$5.0 * 10^{-324}$ - $1.7 * 10^{308}$, 15-digit precision
decimal	16 byte	at least $-7.9 * 10^{?28}$ - $7.9 * 10^{28}$, with at least 28-digit precision

Reference Data Type

The reference data types do not contain the actual data stored in a variable, but they contain a reference to the variables.

If the data is changed by one of the variables, the other variable automatically reflects this change in value.

There are 2 types of reference data type in C# language.

1) Predefined Types - such as Objects, String.

2) User defined Types - such as Classes, Interface.

Pointer Data Type

The pointer in C# language is a variable, it is also known as locator or indicator that points to an address of a value.

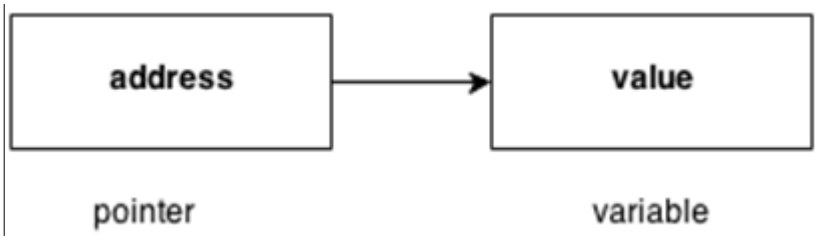
These types of variables are used for storing the address of any memory location which is of another type. Pointers are considered a separate data type kind because they do not hold the actual value or data; rather they are meant to store the actual memory location. The concept of pointers came in C# from C and C++.

The syntax of declaring pointers in C# is:

Syntax:

```
type* identifier;
```

Example:



Symbols used in pointer

Symbol	Name	Description
& (ampersand sign)	Address operator	Determine the address of a variable.
* (asterisk sign)	Indirection operator	Access the value of an address.

Declaring a pointer

The pointer in C# language can be declared using * (asterisk symbol).

```
int * a;  //pointer to int
char * c; //pointer to char
```

ALIAS	TYPE	TYPE NAME	SIZE(BITS)	RANGE	DEFAULT VALUE
Short	signed integer	System.Int16	16	-32768 to 32767	0
Sbyte	signed integer	System.Sbyte	8	-128 to 127	0
Int	signed integer	System.Int32	32	-2^{31} to $2^{31}-1$	0
Long	signed integer	System.Int64	64	-2^{63} to $2^{63}-1$	0L
Byte	unsigned integer	System.byte	8	0 to 255	0
Ushort	unsigned integer	System.UInt16	16	0 to 65535	0

Uint	unsigned integer	System.UInt32	32	0 to 2^{32}	0
Ulong	unsigned integer	System.UInt64	64	0 to 2^{63}	0

Floating point data types in C# are of two types. These are:

1. 32-bit single (7-digit) precision floating point type declared using the keyword float. For initializing any variable with a float, you have to mention a 'f' or 'F' after the value. For example: float g = 62.4f; In case you do not use the suffix, then compiler treats the value as double.
2. 64-bit (14-15 digit) precision floating point type declared using the keyword double. For initializing any variable with double, you have to mention a 'd' or 'D' after the value. For example, double ks =8.403122d;

Decimal Types

There is another type of variable that is used for extensive calculations which are of 128-bit used for calculating huge economic data. It uses' or 'M' as the suffix; otherwise, the value will be treated as double.

Character Types

This is used for representing 16-bit Unicode character used for storing a single character.

Keywords are reserved words having special meaning whose meaning is already defined to the compiler. The keywords in C# are categorized under various groups. These are:

Type	Keywords
Modifier Keywords	abstract, async, const, event, extern, new, override, partial, readonly, sealed, static, unsafe, virtual, volatile

Access Modifier Keywords	public, private, protected, internal
Statement Keywords	if, else, switch, case, do, for, foreach, in, while, break, continue, default, goto, return, yield, throw, try, catch, finally, checked, unchecked, fixed, lock
Method Parameter Keyword	params, ref, out
Access Keywords	base, this
Namespace Keywords	using, . operator, :: operator, extern alias
Literal Keywords	null, false, true, value, void
Operator Keywords	as, await, is, new, sizeof, typeof, stackalloc, checked, unchecked
Contextual Keywords	add, var, dynamic, global, set, value
Type Keywords	bool, byte, char, class, decimal, double, enum, float, int, long, sbyte, short, string, struct, uint, ulong, ushort
Query Keywords	from, where, select, group, into, orderby, join, let, in, on, equals, by, ascending, descending

Some identifiers which have special meaning in context of code are called as **Contextual Keywords.**

add	group	ascending	descending	dynamic	from	get
global	alias	into	join	let	select	set
partial (typ	partial(method)	remove	orderby			

e)				

What Are the Operators in C#?

Operators are symbols in a programming language that tells the compiler or interpreter to perform specific operations on operands for producing the final output or result.

There are six different types of operators provided by C#. These are:

- Arithmetic Operators
- Logical Operators
- Relational Operators

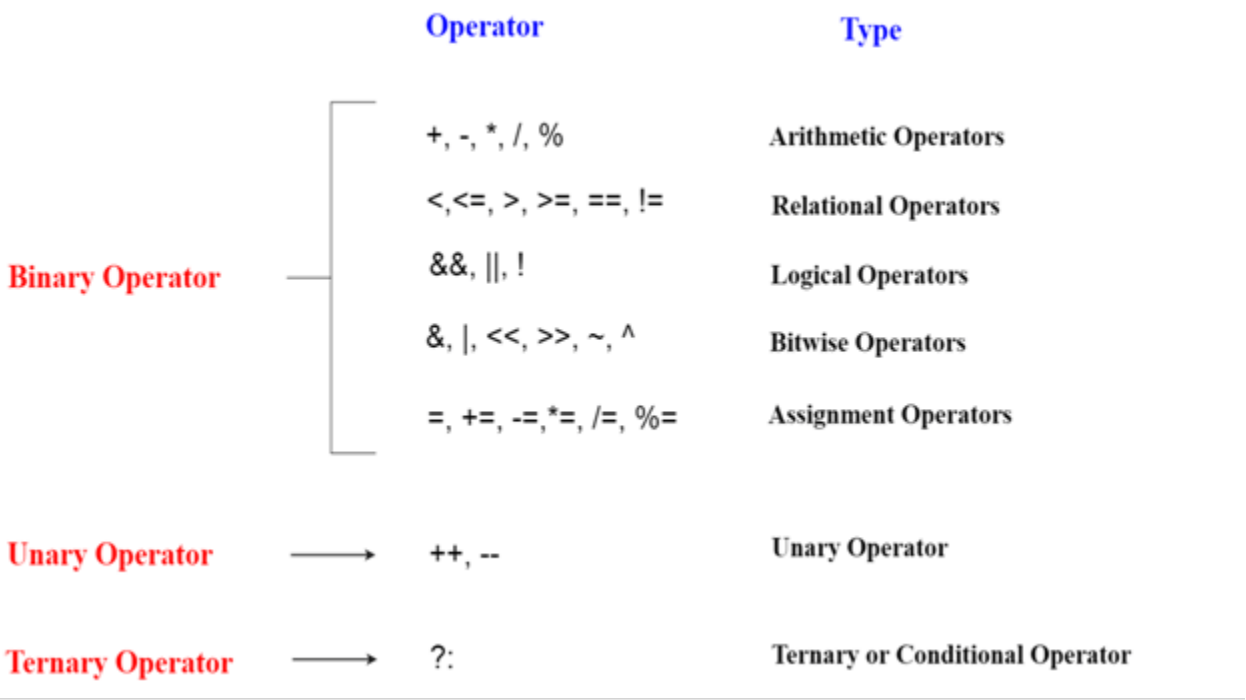
- Assignment Operators
- Bitwise Operators
- Misc Operators

Operator	Description
Addition	The '+' operator is used to add two operands. Like: x + y.
Subtraction	The '-' operator is used to subtract two operands. Like x - y.
Multiplication	The '*' operator is used to multiply two operands. Like: x * y.
Division	The '/' operator is used to divide the first operand by the second one. Like: x / y.
Modulus	The '%' operator is used to return a remainder when the first operand is divided by the second one. Like: x % y.
Increment	The '++' operator is used to increase the integer value by 1. Like: x++ or ++x.
Decrement	The '--' operator is used to decreasing the integer value by 1. Like: x-- or --x.
Operator	Description
Logical AND (&&)	Logical AND (&&) operator returns true if both the conditions/operands satisfy; otherwise returns false.
Logical OR (||)	Logical OR (||) operator returns true if anyone or even both of the conditions/operands satisfy; otherwise returns false.
Logical NOT (!)	Logical NOT (!) operator returns true if the condition is not satisfied; otherwise returns false.
Operator	Description

Equal To operator (==)	Equal To operator (==) operator is used to check if two operands are equal or not. If so, it returns true; otherwise false. Like: 6==6 will return true.
Not Equal To (!=)	Not Equal To (!=) operator is used to checking if two operands are equal or not. If not, it returns true; otherwise false. Like: 6!=2 will return true.
Greater Than (>)	Greater than (>) operator is used to check whether the first operand is larger than the second. If so, returns true; otherwise false. Like: 8> four will return true.
Less Than (<)	Less Than (<) operator is used to check if the first operand is smaller than the second. If so, returns true; otherwise false. Like: 8<1 will return false.
Greater Than Equal To (>=)	Greater Than Equal To (>=) operator is used to check if the first operand is larger than or equal to the second. If so, returns true; otherwise false. Like: 2>=7 will return false.
Less Than Equal To	Less Than Equal To (<=) operator is used to checking if the first operand is smaller than or equal to the second. If so, returns true; otherwise false. Like: 6<=6 will return true.
Operator	Description
Bitwise AND (&)	Bitwise AND (&) operator takes two operands and does AND operation on every bit of those numbers.
Bitwise OR (|)	Bitwise OR (|) operator takes two operands and does OR operation on every bit of those numbers.
Bitwise XOR (^)	Bitwise XOR (^) operator takes two operands and does XOR operation on every bit of those numbers.
Bitwise Left Shift (<<)	Bitwise Left Shift (<<) operator takes two operands and does left shifting of bits on the two operands and determines the number of places to shift.
Bitwise Right Shift (>>)	Bitwise Right Shift (>>) operator takes two operands and does right shifting of bits on the two operands and

	determines the number of places to shift.
Operator	Description
= operator	= operator assigns the value of right side operand to its left side operand. Like: g = s.
+= Operator	+= Operator adds the value of the variable on the left with the value on the right which is (result) then assigned to the variable that is on the left.
-= Operator	-= Operator subtracts the value of the variable on the left with the value on the right which is (result) then assigned to the variable that is on the left.
*= Operator	*= Operator multiplies the value of the variable on the left with the value on the right which is (result) then assigned to the variable that is on the left.
/= Operator	/= Operator divides the value of the variable on the left with the value on the right which is (result) then assigned to the variable that is on the left.
%= Operator	%= Operator first modulo the current value of the variable on the left with the value on the right and (the result) then assigned to the variable that is on the left.
<<= Operator	<<= is the left shift assignment operator that will first left shift the current value of the variable on the left with the value on the right and the result is then assigned to the variable that is on the left.
>>= Operator	>>= is the right shift assignment operator that will first right shift the current value of the variable on the left with the value on the right and the result is then assigned to the variable that is on the left.
&= Operator	&= is the Bitwise AND operator that will first perform a "Bitwise AND" with the current value of the variable on the left with the value on the right, and the result is then assigned to the variable that is on the left.

^= Operator	^= is a Bitwise Exclusive OR operator that will first perform a "Bitwise Exclusive OR" with the current value of the variable on the left with the value on the right and the result is then assigned to the variable that is on the left.
|= Operator	|= is a Bitwise Inclusive OR operator that will first perform a "Bitwise Inclusive OR" with the current value of the variable on the left with the value on the right, and the result is then assigned to the variable that is on the left.
Operator	Description
sizeof()	sizeof() is used to return the data type size.
typeof()	typeof() is used to return the class type.
Pointer to a variable (*) operator	Pointer to a variable (*) operator is used to point to a variable.
Address of (&) operator:	Address of (&) operator is used to return the variable's address.
Conditional operator (?:)	Conditional operator (?:) determines if any condition is true? If yes then first value: otherwise second value.
is:	"is:" finds out if an object is of a certain type or not.



Precedence of Operators in C#

Category (By Precedence)	Operator(s)	Associativity
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to Left
Additive	+ -	Left to Right
Multiplicative	% / *	Left to Right
Relational	< > <= >=	Left to Right
Shift	<< >>	Left to Right
Equality	== !=	Right to Left
Logical AND	&	Left to Right
Logical OR	|	Left to Right
Logical XOR	^	Left to Right
Conditional OR	||	Left to Right

Conditional AND	&&	Left to Right
Null Coalescing	??	Left to Right
Ternary	?:	Right to Left
Assignment	= *= /= %= += - = <<= >>= &= ^= |= =>	Right to Left

C# Control

C# If Example

```
using System;
public class IfExample
   {
     public static void Main(string[] args)
      {
         int num = 10;
         if (num % 2 == 0)
         {
            Console.WriteLine("It is even number");
         }

      }
   }
```

Output:

```
It is even number
```

C# If-else Example

```
using System;
public class IfExample
   {
      public static void Main(string[] args)
      {
         int num = 11;
         if (num % 2 == 0)
         {
            Console.WriteLine("It is even number");
         }
         else
         {
            Console.WriteLine("It is odd number");
         }

      }
   }
```

Output:

```
It is odd number
```

C# If-else Example: with input from user

In this example, we are getting input from the user using **Console.ReadLine()** method. It returns string. For numeric value, you need to convert it into int using **Convert.ToInt32()** method.

```
1.          using System;
2.          public class IfExample
3.            {
4.              public static void Main(string[] args)
5.               {
6.                  Console.WriteLine("Enter a number:");
7.                  int num = Convert.ToInt32(Console.ReadLine());
8.
9.                  if (num % 2 == 0)
10.                 {
11.                    Console.WriteLine("It is even number");
12.                 }
13.                 else
14.                 {
15.                    Console.WriteLine("It is odd number");
16.                 }
17.
18.             }
19.         }
```

Output:

```
Enter a number:11
It is odd number
```

Output:

```
Enter a number:12
It is even number
```

C# IF-else-if ladder Statement

```
1.          using System;
2.          public class IfExample
3.            {
4.               public static void Main(string[] args)
5.               {
6.                  Console.WriteLine("Enter a number to check grade:");
7.                  int num = Convert.ToInt32(Console.ReadLine());
8.
9.                  if (num <0 || num >100)
10.                 {
11.                    Console.WriteLine("wrong number");
12.                 }
13.                 else if(num >= 0 && num < 50){
14.                    Console.WriteLine("Fail");
15.                 }
16.                 else if (num >= 50 && num < 60)
17.                 {
18.                    Console.WriteLine("D Grade");
19.                 }
20.                 else if (num >= 60 && num < 70)
21.                 {
22.                    Console.WriteLine("C Grade");
23.                 }
24.                 else if (num >= 70 && num < 80)
25.                 {
26.                    Console.WriteLine("B Grade");
27.                 }
28.                 else if (num >= 80 && num < 90)
29.                 {
30.                    Console.WriteLine("A Grade");
31.                 }
32.                 else if (num >= 90 && num <= 100)
33.                 {
34.                    Console.WriteLine("A+ Grade");
35.                 }
36.              }
37.           }
```

Output:

```
Enter a number to check grade:66
C Grade
```

Output:

```
Enter a number to check grade:-2
wrong number
```

```
using System;
```

```
namespace Dec_making
{

class IfConditions
{
public static void Main(string[] args)
{
int number = 20;
if (number < 5)
{
Console.WriteLine("{0} is less than 5", number);
}
else
{
    if(number < 10)
        {
        Console.WriteLine("{0} is less than 10", number);
        }
        else
            Console.WriteLine("{0} is greater than 10", number);
}

Console.WriteLine(" Lastly, this statement will be executed.");
}
}

}
```

C# Switch Example

```
1.          using System;
2.           public class SwitchExample
3.            {
4.             public static void Main(string[] args)
5.             {
6.                Console.WriteLine("Enter a number:");
7.                int num = Convert.ToInt32(Console.ReadLine());
8.
9.                switch (num)
10.               {
11.                  case 10: Console.WriteLine("It is 10"); break;
12.                  case 20: Console.WriteLine("It is 20"); break;
13.                  case 30: Console.WriteLine("It is 30"); break;
14.                  default: Console.WriteLine("Not 10, 20 or 30"); break;
15.               }
16.            }
17.           }
```

Output:

```
Enter a number:
10
It is 10
```

Output:

```
Enter a number:
55
Not 10, 20 or 30
```

Note: In C#, break statement is must in switch cases.

```
using System;
namespace NestedSwitch
{

class Program
{
    static void Main(string[] args)
    {
        string title = "student";
        int grade = 1;
        switch( title ) {
            case "teacher":
                Console.WriteLine("You are a teacher.");
                break;
            case "student":
                Console.WriteLine("You are a student.");
                switch( grade ){
                    case 1:
                        Console.WriteLine(" You are in pre-school ");
                        break;
                    case 2:
                        Console.WriteLine(" You are in high-school ");
                        break;
                    case 3:
                        Console.WriteLine(" You are in college ");
                        break;
                    case 4:
                        Console.WriteLine(" You are in university ");
                        break;
                    default:
                        Console.WriteLine("Result Unknown");
                        break;
                }
                break;
            default:
                Console.WriteLine(" Office Staff ");
                break;
        }
        Console.Read();
    }
}

}
```

10._Infinite Loops

What Are Loops in C#?

Loop is a concept that is used in almost all programming language for executing a single statement or collection of statements several times depending on the condition.

The loops in C# are basically of two types depending on their behavior. These are:

- Entry controlled loop
- Exit controlled loop

C# For Loop Example

```
1.          using System;
2.          public class ForExample
3.            {
4.              public static void Main(string[] args)
5.              {
6.                 for(int i=1;i<=10;i++){
7.                   Console.WriteLine(i);
8.                 }
9.              }
10.           }
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

nested for loop in C#.

```
1.          using System;
2.          public class ForExample
3.            {
4.              public static void Main(string[] args)
5.              {
6.                for(int i=1;i<=3;i++){
7.                      for(int j=1;j<=3;j++){
8.                         Console.WriteLine(i+" "+j);
9.                      }
10.               }
11.             }
12.           }
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
```

```
3 3
```

C# Infinite For Loop

```
1.          using System;
2.          public class ForExample
3.             {
4.               public static void Main(string[] args)
5.               {
6.                  for (; ;)
7.                  {
8.                        Console.WriteLine("Infinitive For Loop");
9.                  }
10.              }
11.          }
```

Output:

```
Infinitive For Loop
Infinitive For Loop
Infinitive For Loop
Infinitive For Loop
Infinitive For Loop
```

C# While Loop

C# While Loop Example

```
1.          using System;
2.          public class WhileExample
3.             {
4.               public static void Main(string[] args)
5.               {
6.                  int i=1;
7.                  while(i<=10)
8.                  {
9.                     Console.WriteLine(i);
10.                    i++;
11.                 }
12.              }
13.            }
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

C# Nested While Loop Example:

```
using System;
public class WhileExample
   {
     public static void Main(string[] args)
     {
        int i=1;
        while(i<=3)
        {
           int j = 1;
           while (j <= 3)
           {
              Console.WriteLine(i+" "+j);
              j++;
           }
           i++;
        }
     }
   }
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

C# Infinitive While Loop Example:

```
1.          using System;
2.          public class WhileExample
3.             {
4.               public static void Main(string[] args)
5.               {
6.                  while(true)
7.                  {
8.                        Console.WriteLine("Infinitive While Loop");
9.                  }
10.              }
11.            }
```

Output:

```
Infinitive While Loop
Infinitive While Loop
Infinitive While Loop
Infinitive While Loop
Infinitive While Loop
```

C# do-while Loop Example

Let's see a simple example of C# do-while loop to print the table of 1.

```
1.          using System;
2.          public class DoWhileExample
3.             {
4.               public static void Main(string[] args)
5.               {
6.                  int i = 1;
7.
8.                  do{
9.                     Console.WriteLine(i);
10.                    i++;
11.                 } while (i <= 10) ;
12.
13.              }
14.            }
```

Output:

```
1
2
```

```
3
4
5
6
7
8
9
10
```

C# Nested do-while Loop

```
using System;
public class DoWhileExample
   {
     public static void Main(string[] args)
     {
        int i=1;

        do{
           int j = 1;

           do{
              Console.WriteLine(i+" "+j);
              j++;
           } while (j <= 3) ;
           i++;
        } while (i <= 3) ;
     }
    }
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

C# Infinitive do-while Loop Example

```
1.          using System;
2.          public class WhileExample
3.             {
4.               public static void Main(string[] args)
5.               {
6.
7.                  do{
8.                     Console.WriteLine("Infinitive do-while Loop");
9.                  } while(true);
10.              }
11.            }
```

Output:

```
Infinitive do-while Loop
Infinitive do-while Loop
Infinitive do-while Loop
Infinitive do-while Loop
Infinitive do-while Loop
ctrl+c
```

Jump Statements in C#

Jumping of execution is another important concept of programming, and the different jump statements help us to do so. These statements help in transferring control from one point to the other. Positioning the execution based on a certain requirement is another concept that helps in jumping from a particular logic to others within a program.

C# provides programmers with five different types of jump statements. These are:

1. goto
2. break
3. continue
4. return
5. throw

C# Goto Statement

The C# goto statement is also known jump statement. It is used to transfer control to the other part of the program. It unconditionally jumps to the specified label.

It can be used to transfer control from deeply nested loop or switch case label.

Currently, it is avoided to use goto statement in C# because it makes the program complex.

```
1.  using System;
2.  public class GotoExample
3.     {
4.       public static void Main(string[] args)
5.       {
6.       ineligible:
7.          Console.WriteLine("You are not eligible to vote!");
8.
9.       Console.WriteLine("Enter your age:\n");
10.      int age = Convert.ToInt32(Console.ReadLine());
11.      if (age < 18){
12.            goto ineligible;
13.      }
14.      else
15.      {
16.            Console.WriteLine("You are eligible to vote!");
17.      }
18.      }
19.   }
```

Output:

```
You are not eligible to vote!
```

```
Enter your age:
11
You are not eligible to vote!
Enter your age:
5
You are not eligible to vote!
Enter your age:
26
You are eligible to vote!
```

C# Break Statement

The C# *break* is used to break loop or switch statement. It breaks the current flow of the program at the given condition. In case of inner loop, it breaks only inner loop.

```
1.  using System;
2.  public class BreakExample
3.    {
4.      public static void Main(string[] args)
5.      {
6.         for(int i=1;i<=3;i++){
7.               for(int j=1;j<=3;j++){
8.                  if(i==2&&j==2){
9.                     break;
10.                 }
11.                 Console.WriteLine(i+" "+j);
12.              }
13.        }
14.     }
15.   }
```

Output:

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

C# Continue Statement

The C# *continue statement* is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

```
1.  using System;
2.  public class ContinueExample
```

```
3.          {
4.            public static void Main(string[] args)
5.            {
6.               for(int i=1;i<=3;i++){
7.                     for(int j=1;j<=3;j++){
8.                        if(i==2&&j==2){
9.                           continue;
10.                       }
11.                       Console.WriteLine(i+" "+j);
12.                    }
13.              }
14.           }
15.         }
```

Output:

```
1 1
1 2
1 3
2 1
2 3
3 1
3 2
3 3
```

return

This jump statement is used to terminate any execution flow in any method and returns the control to calling method. The value returned by this is optional and when the return type is void, the return statement can be debarred.

Example:

```
using System;

class ProgEg {

static int Add(int aa)

{

    int a = aa + aa;

    return a;

}

static public void Main()

{
```

```
    int numb = 6;
    int res = Add(numb);
    Console.WriteLine(" Addition is: {0} : ", res);
}
}
```

Output:

```
Addition is: 12
```

throw

This is a concept of exception handling in C# which will be covered in the later chapters. But since it also jumps program flows and executions by creating an object of valid exception class using the *new*

Example:

```
using System;
class ProgEg
 {
    static string s = null;
    static void disp(string s1)
    {
        if (s1  == null)
            throw new NullReferenceException("Exception Found.");
    }
static void Main(string[] args)
{
    try
    {
        disp(s);
    }
```

```
    catch(Exception exp)
    {
        Console.WriteLine( exp.Message );
    }
}
}
```

Output:

```
Exception Found.
```

The exception thrown by the throw statement will be taken care of by the catch statement which jumps the execution flow directly from throw to catch.

C# Function