C# | Namespaces

Namespaces are used to organize the classes. It helps to control the scope of methods and classes in larger **.Net** programming projects. In simpler words you can say that it provides a way to keep one set of names(like class names) different from other sets of names. The biggest advantage of using namespace is that the class names which are declared in one namespace will not clash with the same class names declared in another namespace. It is also referred as **named group of classes** having common features. The members of a namespace can be **namespaces, interfaces, structures, and delegates.**

Defining a Namespace

To define a namespace in C#, we will use the **namespace** keyword followed by the name of the namespace and curly braces containing the body of the namespace as follows:
Syntax:

```
namespace name_of_namespace {


// Namespace (Nested Namespaces)

// Classes

// Interfaces

// Structures

// Delegates


}
```

Example:

```
// defining the namespace name1

namespace name1

{


    // C1 is the class in the namespace name1

    class C1

    {
        // class code

    }

}
```

Accessing the Members of Namespace

The members of a namespace are accessed by using dot(.) operator. A class in C# is fully known by its respective namespace.

Syntax:

```
[namespace_name].[member_name]
```

Note:

- Two classes with the same name can be created inside 2 different namespaces in a single program.
- Inside a namespace, no two classes can have the same name.
- In C#, the full name of the class starts from its namespace name followed by *dot(.)* operator and the class name, which is termed as the fully qualified name of the class.

Example:

filter_none
edit
play_arrow
brightness_4

```
// C# program to illustrate the
// use of namespaces

// namespace declaration
namespace first {

    // name_1 namespace members
    // i.e. class
    class Geeks_1
    {

        // function of class Geeks_1
        public static void display()
        {
            // Here System is the namespace
            // under which Console class is defined
            // You can avoid writing System with
            // the help of "using" keyword discussed
            // later in this article
            System.Console.WriteLine("Hello Geeks!");

        }
    }
    /* Removing comment will give the error
       because no two classes can have the
       same name under a single namespace

    class Geeks_1
    {

    }       */


} // ending of first namespace


// Class declaration
class Geeks_2
{

    // Main Method
    public static void Main(String []args)
    {

        // calling the display method of
        // class Geeks_1 by using two dot
        // operator as one is use to access
        // the class of first namespace and
        // another is use to access the
        // static method of class Geeks_1.
        // Termed as fully qualified name
        first.Geeks_1.display();

    }
}
```

Output:

```
Hello Geeks!
```

In the above example:

- In **System.Console.WriteLine()**" "*System*" is a namespace in which we have a class named "*Console*" whose method is "*WriteLine()*".
- It is not necessary to keep each class in C# within Namespace but we do it to organize our code well.
- Here "**.**" is the delimiter used to separate the class name from the namespace and function name from the classname.

The using keyword

It is not actually practical to call the function or class(or you can say members of a namespace) every time by using its fully qualified name. In the above example, *System.Console.WriteLine("Hello Geeks!");* and *first.Geeks_1.display();* are the fully qualified name. So C# provides a keyword "**using**" which help the user to avoid writing fully qualified names again and again. The user just has to mention the namespace name at the starting of the program and then he can easily avoid the use of fully qualified names.

Syntax:

```
using [namespace_name][.][sub-namespace_name];
```

In the above syntax, *dot(.)* is used to include subnamespace names in the program.

Example:

```
// predefined namespace name

using System;



// user-defined namespace name

using name1



// namespace having subnamespace

using System.Collections.Generic;
```

Program:

filter_none
edit
play_arrow
brightness_4

```
// C# program to illustrate the
// use of using keyword

// predefined namespace
using System;

// user defined namespace
using first;

// namespace declaration
namespace first {

    // name_1 namespace members
    // i.e. class
    class Geeks_1
    {

        // function of class Geeks_1
        public static void display()
        {
            // No need to write fully qualified name
            // as we have used "using System"
            Console.WriteLine("Hello Geeks!");

        }
    }


} // ending of first namespace


// Class declaration
class Geeks_2
{

    // Main Method
    public static void Main(String []args)
    {

        // calling the display method of
        // class Geeks_1 by using only one
        // dot operator as display is the
        // static method of class Geeks_1
        Geeks_1.display();

    }
}
```

Output:

```
Hello Geeks!
```

Nested Namespaces

You can also define a namespace into another namespace which is termed as the nested namespace. To access the members of nested namespace user has to use the dot(.) operator.

For example, Generic is the nested namespace in the *collections* namespace as *System.Collections.Generic*

Syntax:

```
namespace name_of_namespace_1

{



   // Member declarations & definitions

   namespace name_of_namespace_2

   {
        // Member declarations & definitions
        .
        .

   }
}
```

Program:

filter_none
edit
play_arrow
brightness_4

```
// C# program to illustrate use of
// nested namespace
using System;

// You can also use
// using Main_name.Nest_name;
// to avoid the use of fully
// qualified name

// main namespace
namespace Main_name
{

    // nested namespace
    namespace Nest_name
    {

        // class within nested namespace
        class Geeks_1
         {

              // Constructor of nested
              // namespace class Geeks_1
              public Geeks_1() {

                  Console.WriteLine("Nested Namespace Constructor");

              }
        }
    }
}

// Driver Class
class Driver
```

```
{

    // Main Method
    public static void Main(string[] args)
    {

        // accessing the Nested Namespace by
        // using fully qualified name
        // "new" is used as Geeks_1()
        // is the Constructor
        new Main_name.Nest_name.Geeks_1();

    }
}
```

Output:

```
Nested Namespace Constructor
```

Using Directive

Generally, we use the using keyword to add namespaces in code-behind and class files. Then it makes all the classes, interfaces and abstract classes and their methods and properties available in the current page. Adding a namespace can be done in the following two ways,

A. To allow the normal use of types in a namespace,

```
using System.IO;
using System.Text;
```

B. To create an alias for a namespace or a type. This is called a using alias directive.

```
using MyProject = TruckingApp.Services;
```

We can use the namespace alias as in the following:

```
MyProject.Truck newObj = new MyProject.Truck();
```

This one (option B) is very useful when the same class/abstract/interface is present in multiple namespaces.

Let's say the Truck class is present in TruckingApp1, TruckingApp2, and TruckingApp3. Then it is difficult to call the Truck class of namespace2.

Here the alias directive gives an elegant syntax to use the Truck class.

Code

```csharp
using namespace1 = TruckingApp1.Services;
using namespace2 = TruckingApp2.Services;
using namespace3 = TruckingApp3.Services;

namespace2.Truck newObj = new namespace2.Truck();
```

Now the code looks more elegant and easy to understand. Except for this way, you can also access the Truck class using namespace directly as in the following:

```csharp
TruckingApp.Services2.Truck newObj = new TruckingApp.Services2.Truck();
```

This one has one disadvantage. If I am using the Truck class in 100 places, then I need to use the namespace name every time. So always use the alias directive in this scenario.

Using Statement

This is another way to use the using keyword in C#. It plays a vital role in improving performance in Garbage Collection.

Introduction to C#

C# is a simple and powerful programming language which is used to develop many robust applications. It is pronounced as ***C Sharp***.

C# is supported by a large number of frameworks. C# can be used to develop a lot of web-based applications with the .NET Framework which is a software development platform developed by Microsoft. There are other frameworks like Mono or Xamarin which can be used to develop mobile apps, games or iOS and Android applications.

There are many features of C# which have made it one of the most widely used programming languages. It is an object-oriented language which is compatible with other .NET languages. It is type safe because a C# code can access only that part of memory which it has permission to access and execute, thus increasing the safety of the program.

Applications of C#

C# is used to develop a wide range of applications. Following are some of the applications developed using C# :

- **Windows Client Applications** → It is used to build Windows client applications using Windows Forms and WPF, which are application templates provided by Microsoft Visual Studio.
- **Web Applications** → It is used to build modern web applications using ASP.NET combined with JavaScript and other libraries and APIs. ASP.NET is one of the most widely used technology for building web applications and is supported by templates developed by Microsoft Visual Studio.
- **Console Applications** → It is used to build applications that run in a command-line interface.
- **Azure Cloud Applications** → It is used to build cloud-based applications for Windows Azure, which is an operating system of Microsoft for cloud computing and hosting.
- **iOS and Android Mobile Apps** → It is used for cross-platform native mobile app development via Xamarin, which can be used to create platform-specific UI code layer.
- **Game Development** → It is a quite popular language to build games because it is fast and has much control over memory management. It has a rich library for designing graphics. C# is used in many game engines like Unity Engine and Unreal.
- **Internet of Things Devices** → It is used for building IoT devices because it doesn't need much processing power.
- **Artificial Intelligence** → C# has extensive libraries for basic deep learning and embedded systems. As a result, it is used in the fields of Computer Vision and Artificial Intelligence.

How to start?

In this entire C# tutorial, you will be learning about writing C# programs. But what after that?

After writing any program, we need to **compile** and **run** it to see the output of the program.

So, what is meant by compiling?

When we write our program in C# language, it needs to be converted to machine language (which is binary language consisting of 0s and 1s) so that computer can understand and execute it. This conversion is known as compiling a program. We compile a code with the help of a **compiler**.

Integrated Development Environment (IDE)

To write and compile our C# program, we have been provided with many IDEs.

An IDE consists of a **Text Editor** as well as a **Compiler**. We type our program in the text editor which is then compiled by the compiler.

Text Editor

We write our program in a text editor. The file containing our program is called **source file** and is saved with a **.cs** extension.

C# Compiler

After saving our program in a file with the .cs extension, we need to compile it to convert it into machine language that computer can understand.

Now let's see how to compile and execute a C# program on different operating systems. We are only going to show how to run a C# program, the working of the code is explained in the next chapter.

Windows

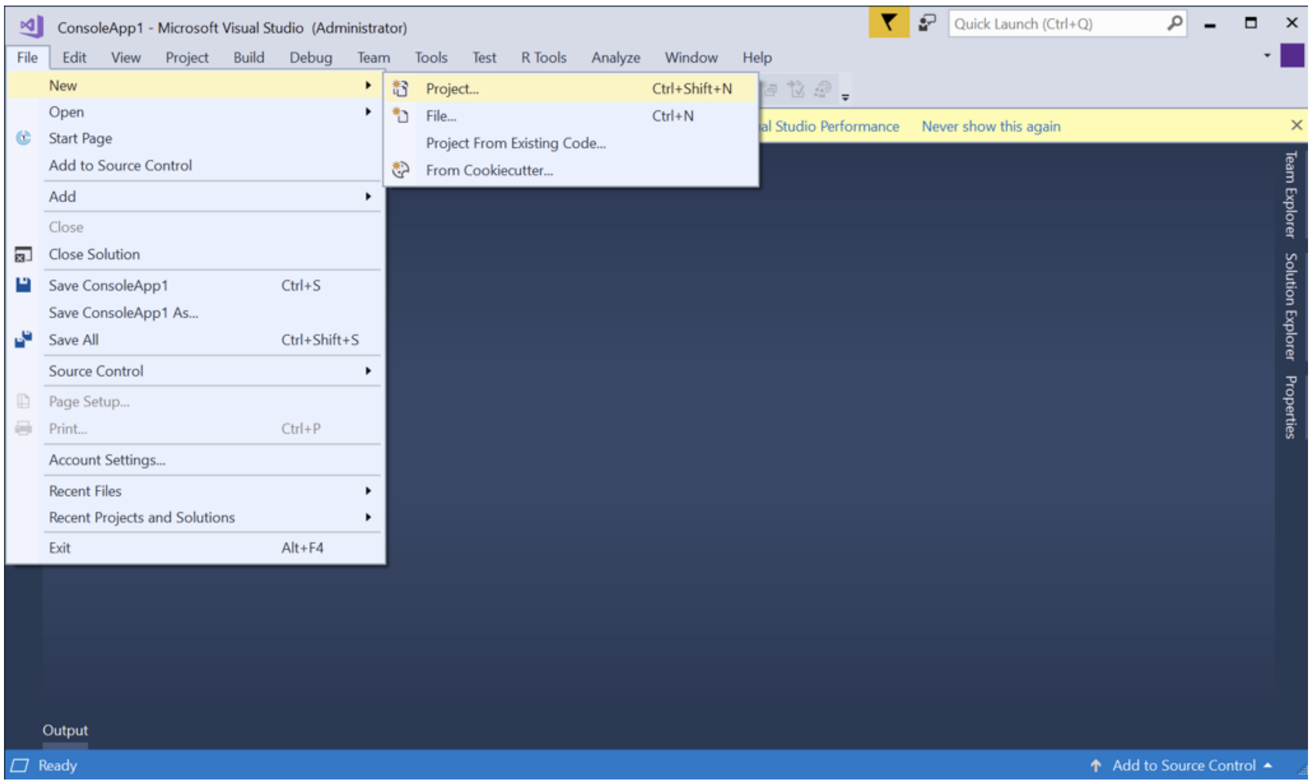
There are many IDEs available for editing and compiling C# programs in Windows like Microsoft Visual Studio and NetBeans. We will see how to run a C# code using Visual Studio and Command Prompt.

Visual Studio

We can edit and compile programs using Visual Studio which allows us to write, compile and run our program. Let's see how to do it.

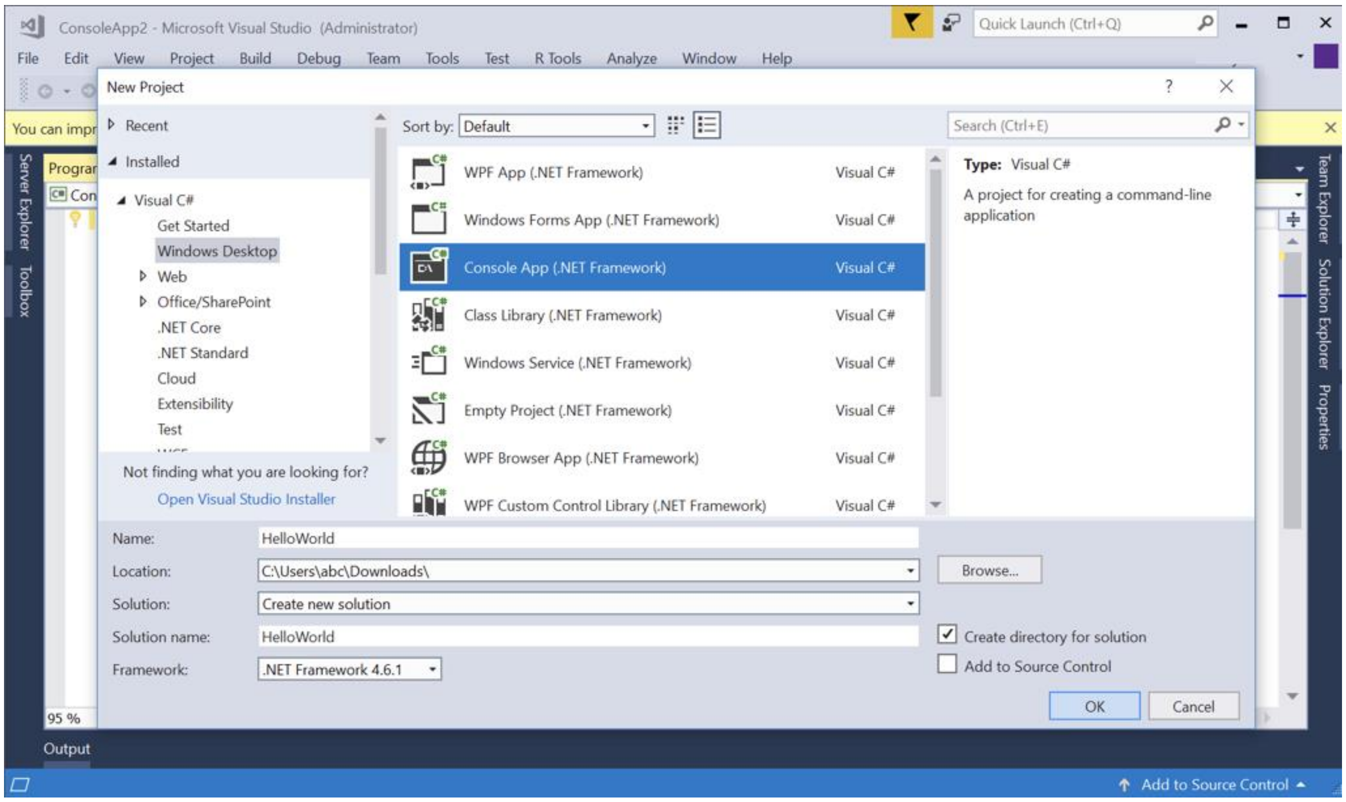
Before learning to write a program in Visual Studio, first, make sure it is installed in your computer. If not, then download and install it from here.

1. On opening Visual Studio, you will get a window. Click on File → New → Project.



2. The New Project dialog box will appear. Expand **Installed**, then expand **Visual C#**, then select **Windows Desktop**, and then select **Console Application**.

3. Give the project name in the **Name** text box, and then click on OK button.


ConsoleApp2 - Microsoft Visual Studio (Administrator)
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Test R Tools Analyze Window Help
New Project
Recent
Installed
Visual C#
Get Started
Windows Desktop
Web
Office/SharePoint
.NET Core
.NET Standard
Cloud
Extensibility
Test
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
WPF App (.NET Framework) Visual C#
Windows Forms App (.NET Framework) Visual C#
Console App (.NET Framework) Visual C#
Class Library (.NET Framework) Visual C#
Windows Service (.NET Framework) Visual C#
Empty Project (.NET Framework) Visual C#
WPF Browser App (.NET Framework) Visual C#
WPF Custom Control Library (.NET Framework) Visual C#
Browse...
OK
Cancel
Server Explorer
Toolbox
Team Explorer
Solution Explorer
Properties
Output
Add to Source Control

Type: Visual C#

A project for creating a command-line application

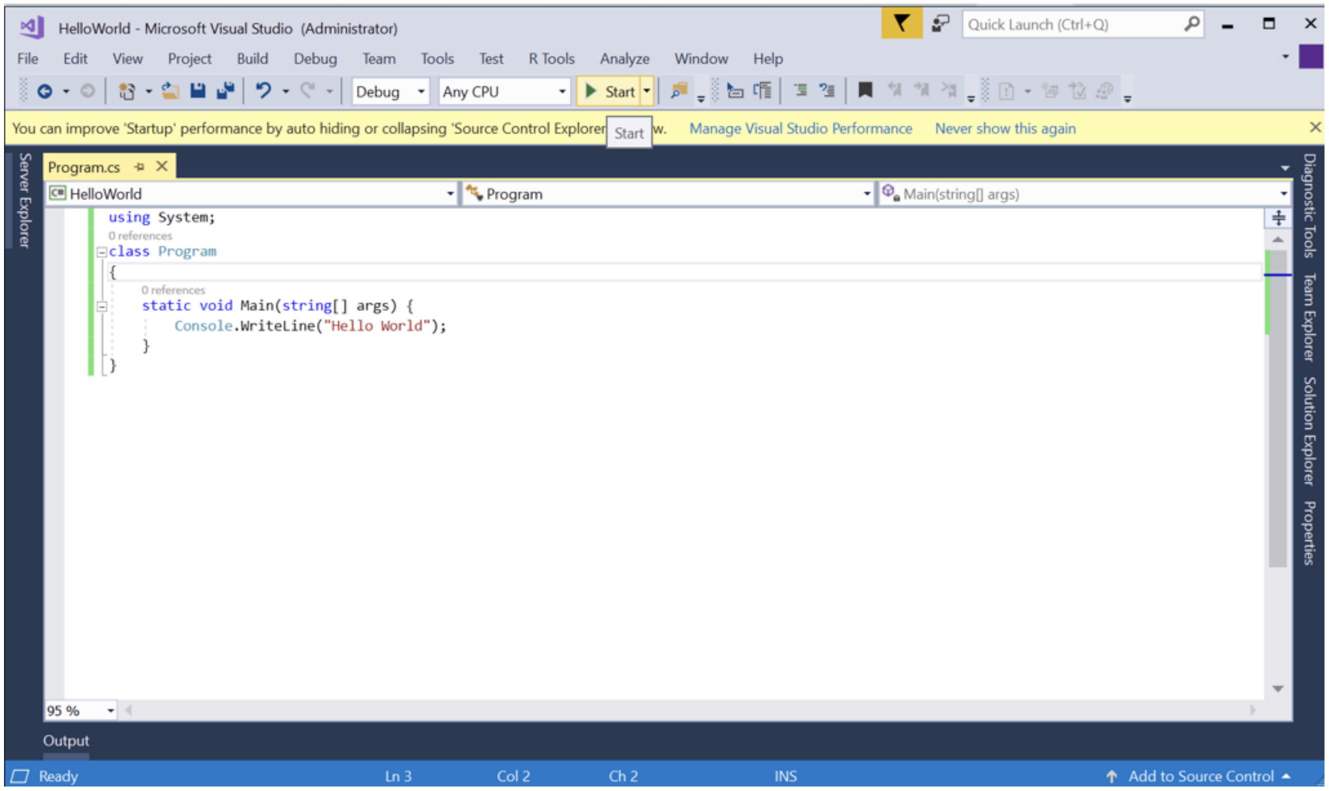
Name:	HelloWorld
Location:	C:\Users\abc\Downloads\
Solution:	Create new solution
Solution name:	HelloWorld
Framework:	.NET Framework 4.6.1

✓ Create directory for solution

☐ Add to Source Control

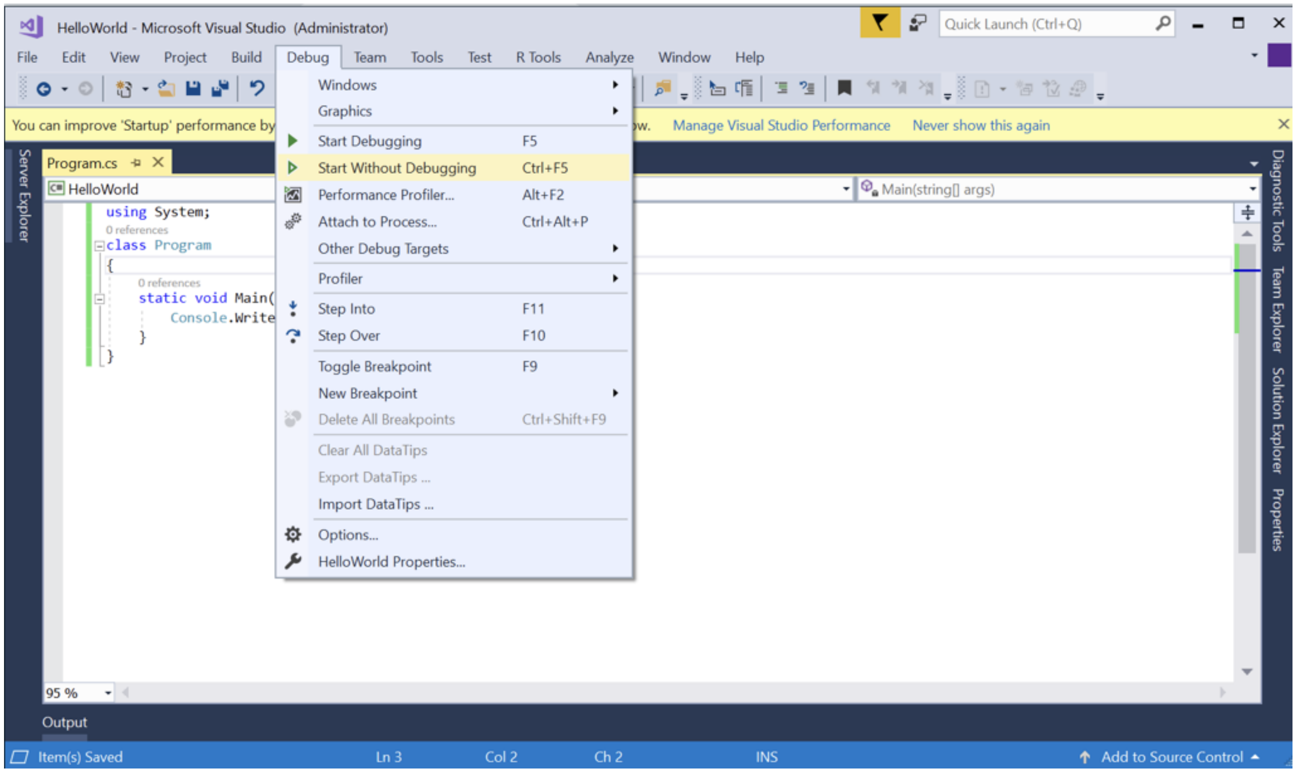
3. A file named *Program.cs* will get opened in the Code Editor. Replace its content with your C# program given below.

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

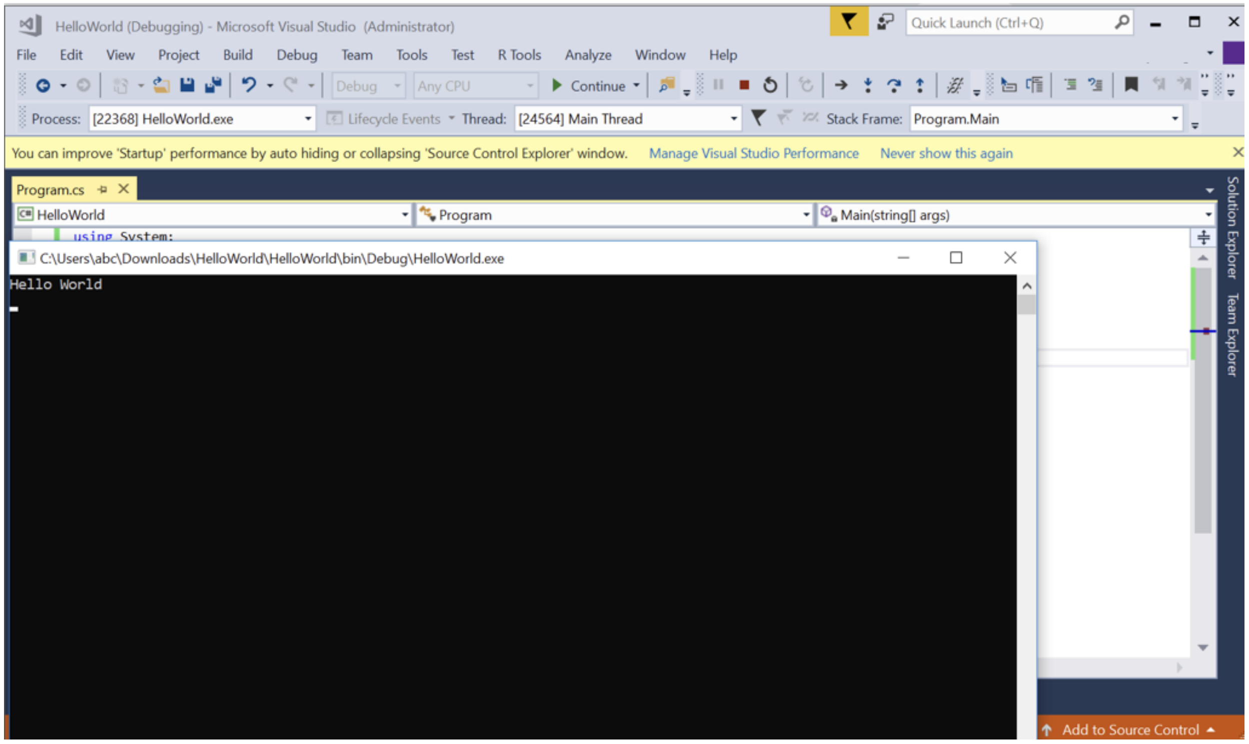


```
using System;
0 references
class Program
{
    0 references
    static void Main(string[] args) {
        Console.WriteLine("Hello World");
    }
}
```

4. To run the project, click on Debug → Start Without Debugging or press `Ctrl+F5` keys.

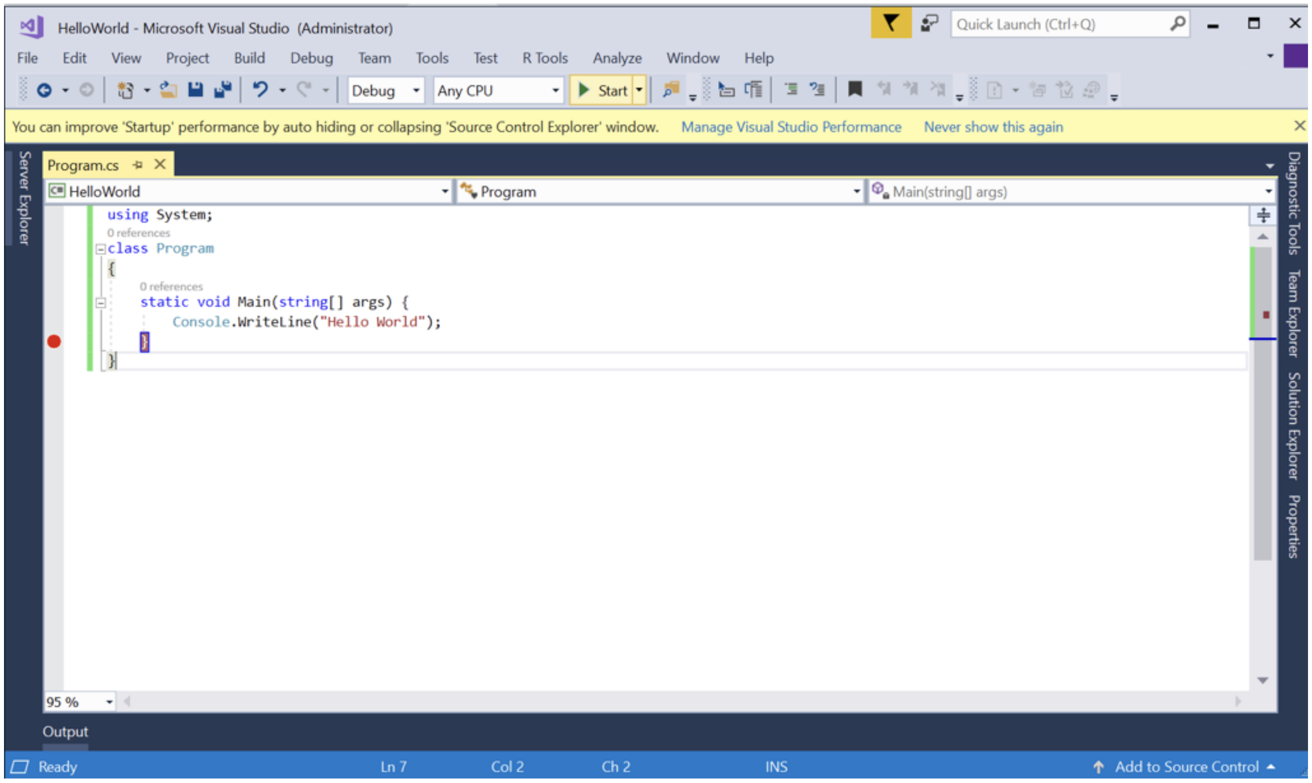


5. A Command Prompt window will appear with "Hello World" written.


C:\Users\abc\Downloads\HelloWorld\HelloWorld\bin\Debug\HelloWorld.exe

```
Hello World
```

If you want to use the debugger, then put the breakpoint on the last line as shown below. Then to run the project, click on the **Start** button or press the **F5** key.



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

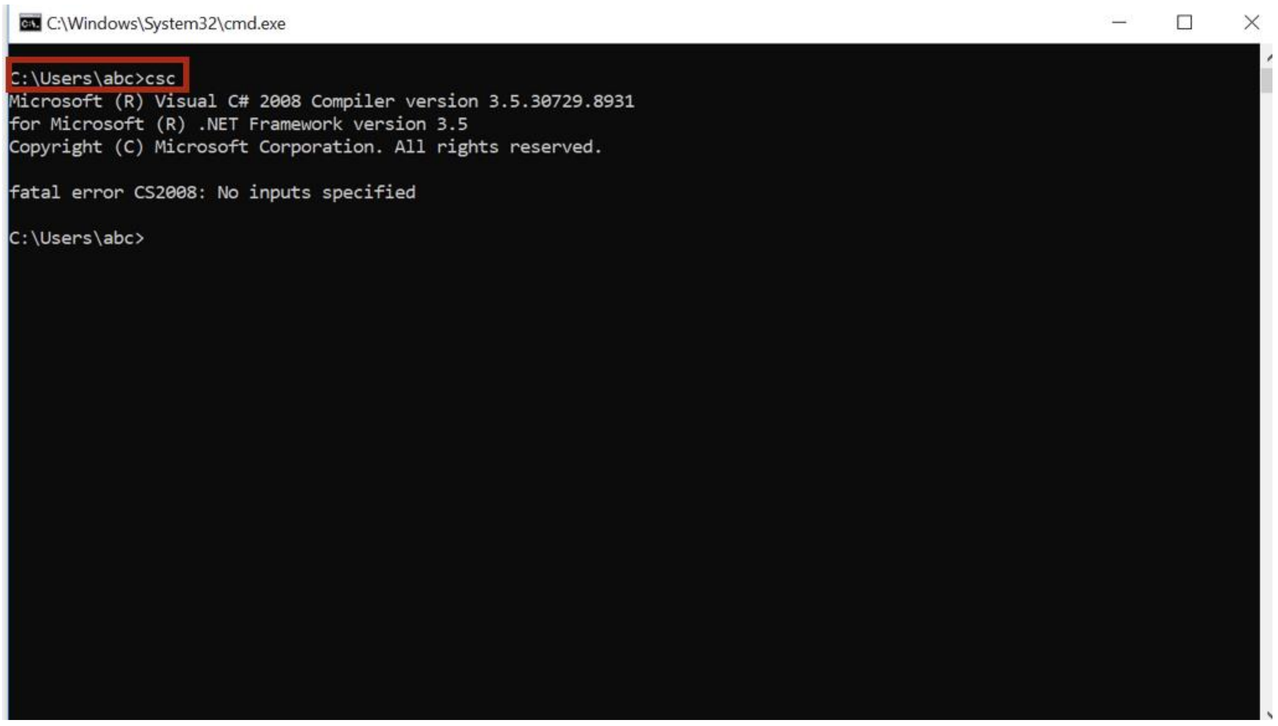
Command Prompt

1. Write your C# program (code given below) in a text editor like Notepad and save it with a filename having `.cs` extension. In our case, we named it as *hello.cs*.

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

2. Open the Command Prompt window (search for **cmd**).

3. To check if the path of the C# compiler is set as Environment Variable, type `csc` and press Enter. You should get something as shown below.

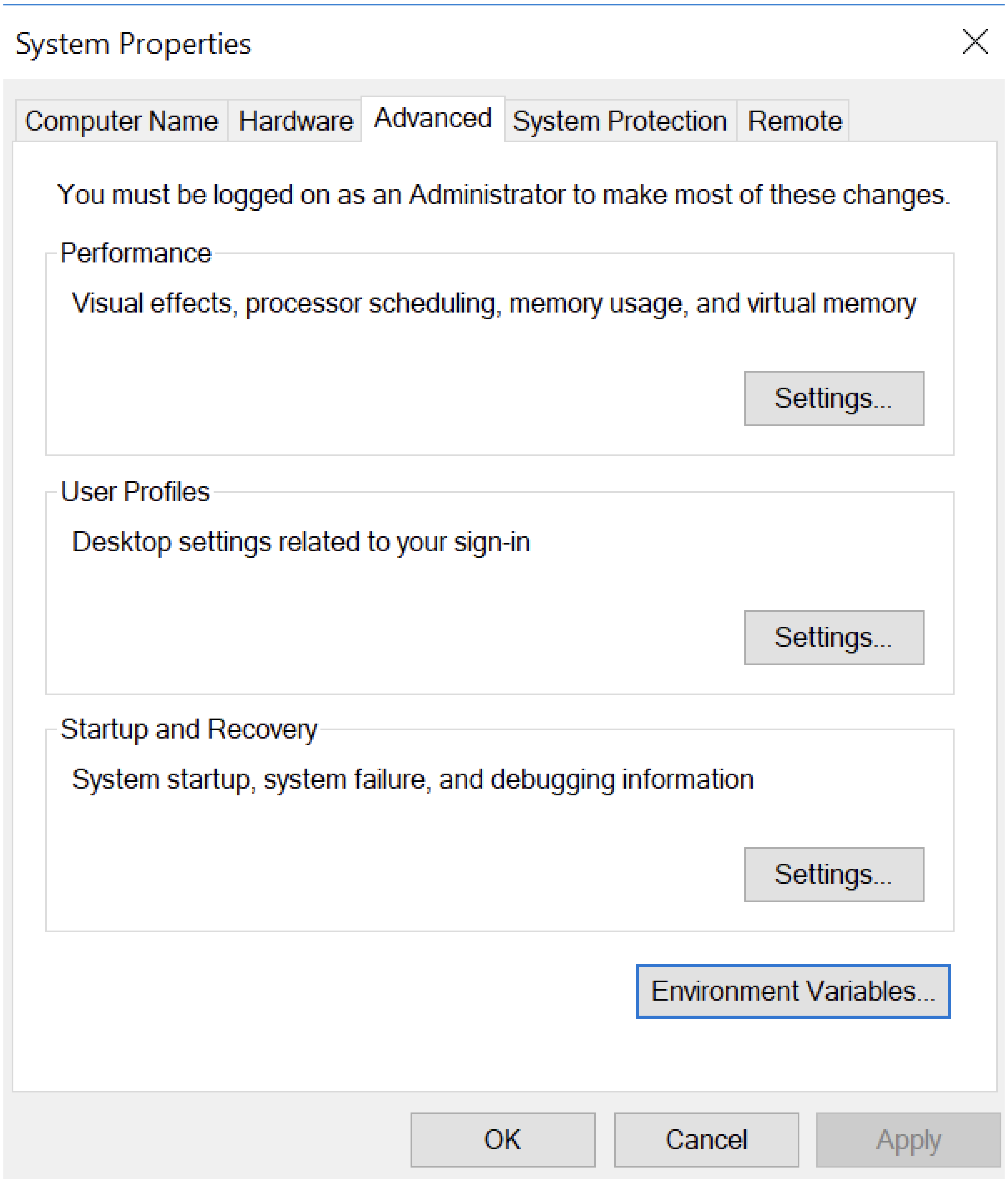


```
C:\Users\abc>csc
Microsoft (R) Visual C# 2008 Compiler version 3.5.30729.8931
for Microsoft (R) .NET Framework version 3.5
Copyright (C) Microsoft Corporation. All rights reserved.

fatal error CS2008: No inputs specified

C:\Users\abc>
```

If you are getting this message - *'csc' is not recognized as an internal or external command, operable program or batch file*, then you need to the path of the C# compiler as Environment Variable.

4. To set the path as Environment Variable, right click on My Computer and select Properties.

5. Go to the Advanced System Settings tab.



System Properties

You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles

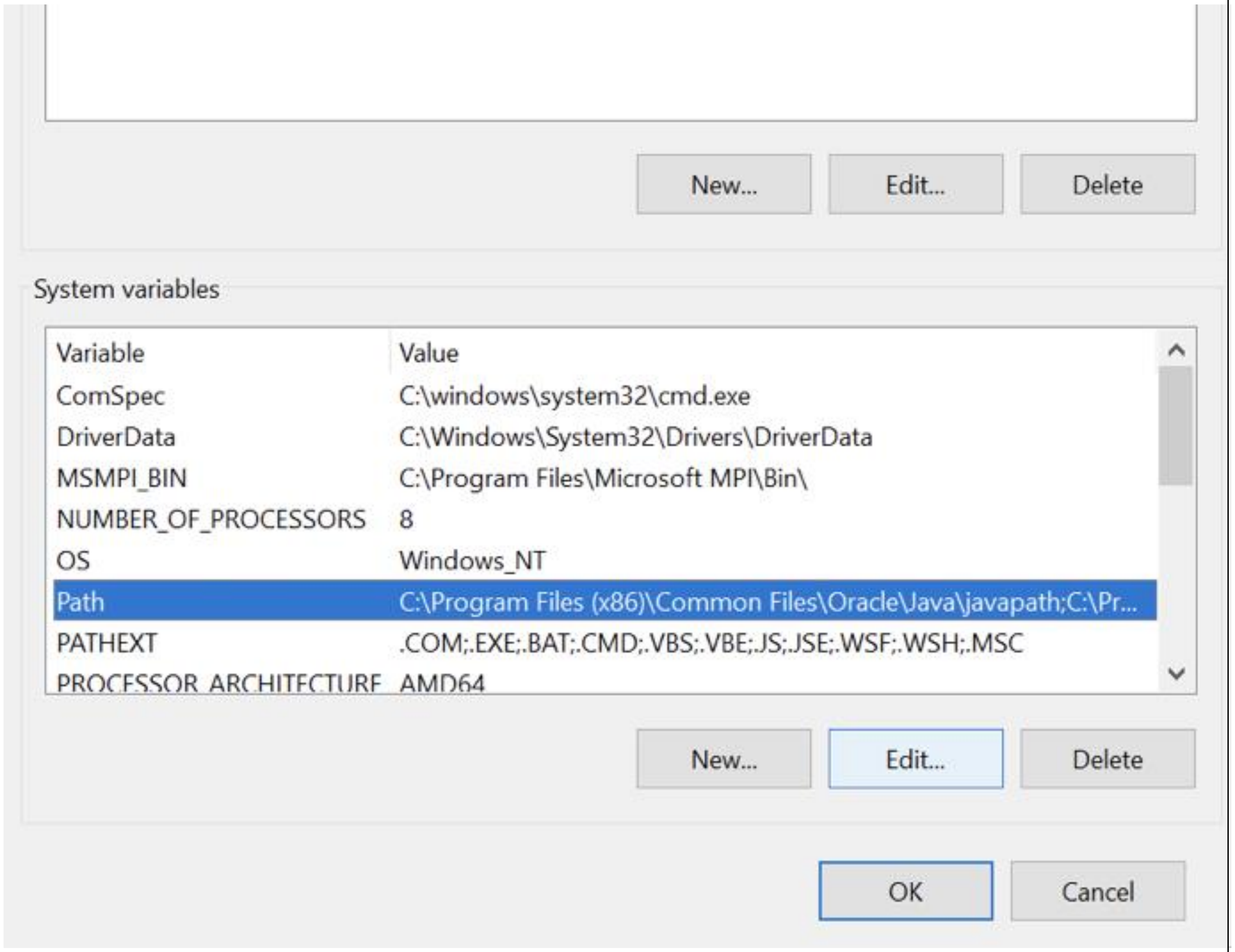
Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information

6. Click on the Environment Variables button.

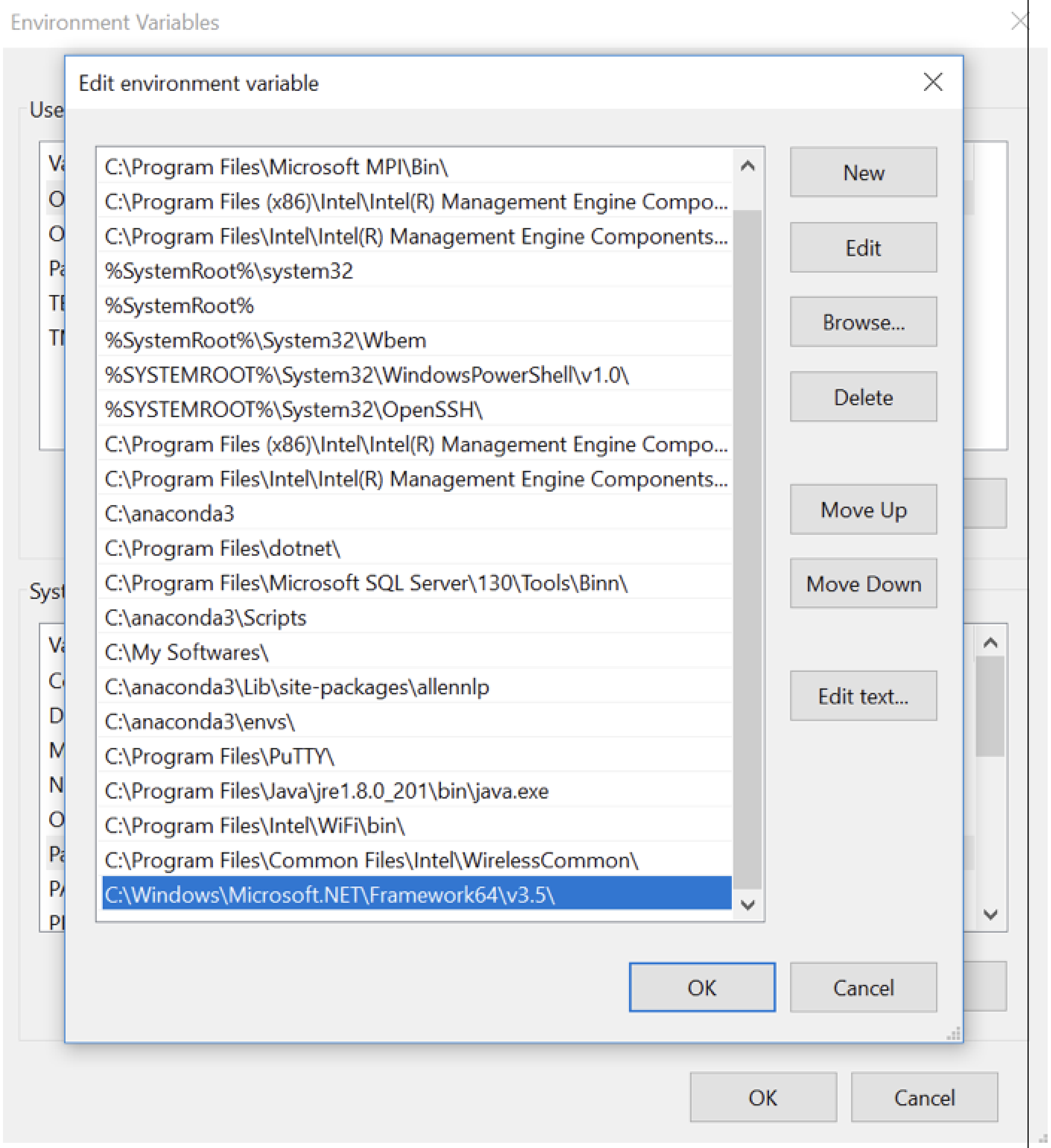
7. In the System Variables box, select **Path** and click on the Edit button.



Variable	Value
ComSpec	C:\windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
MSMPI_BIN	C:\Program Files\Microsoft MPI\Bin\
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Pr...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

8. An Edit Environment variable window will appear. Let's assume that **csc.exe** is located in C:\Windows\Microsoft.NET\Framework64\v3.5 directory. This directory is the path to the compiler.

Click on the New button and add the compiler path in the text box created as shown below. Then click on Ok.



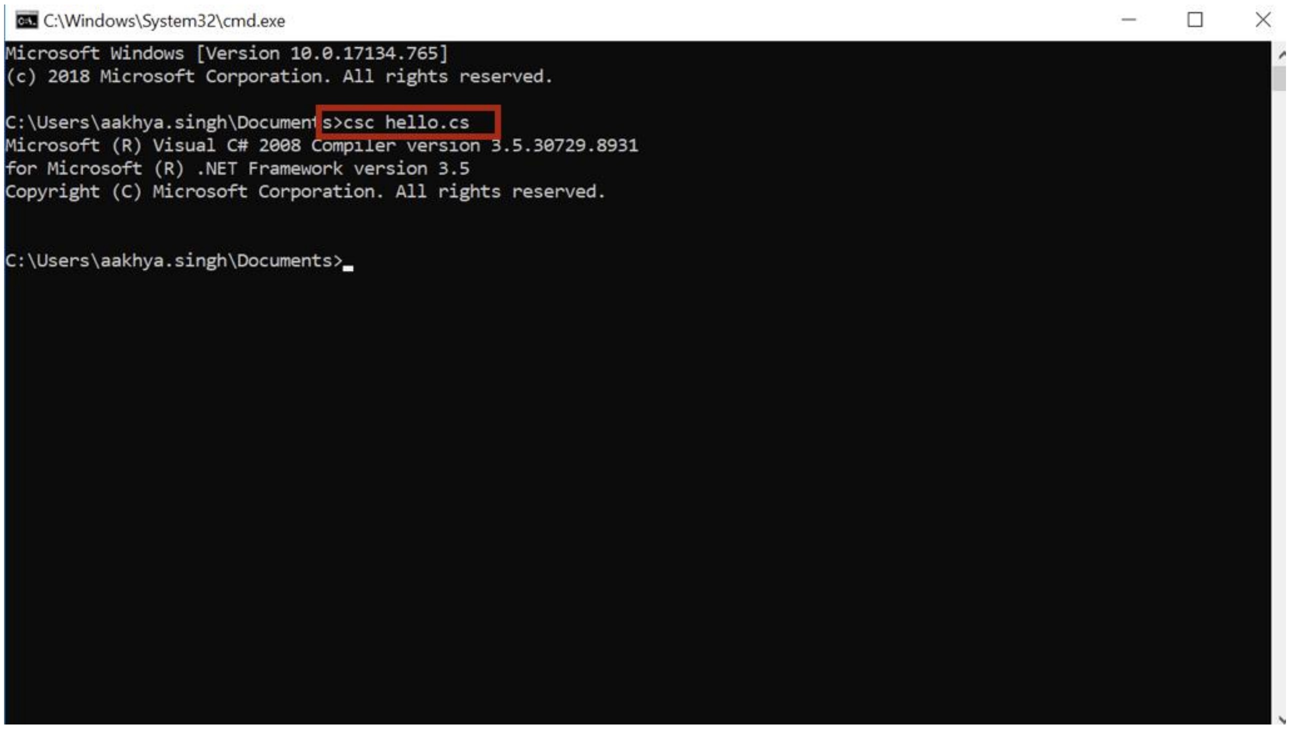
```
C:\Program Files\Microsoft MPI\Bin\
C:\Program Files (x86)\Intel\Intel(R) Management Engine Compo...
C:\Program Files\Intel\Intel(R) Management Engine Components...
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files (x86)\Intel\Intel(R) Management Engine Compo...
C:\Program Files\Intel\Intel(R) Management Engine Components...
C:\anaconda3
C:\Program Files\dotnet\
C:\Program Files\Microsoft SQL Server\130\Tools\Binn\
C:\anaconda3\Scripts
C:\My Softwares\
C:\anaconda3\Lib\site-packages\allennlp
C:\anaconda3\envs\
C:\Program Files\PuTTY\
C:\Program Files\Java\jre1.8.0_201\bin\java.exe
C:\Program Files\Intel\WiFi\bin\
C:\Program Files\Common Files\Intel\WirelessCommon\
C:\Windows\Microsoft.NET\Framework64\v3.5\
```

9. Now open a new Command Prompt window. Use the command **cd** followed by a directory name to change your working directory. For example, if you are operating in C:\Users\John\Project and want to get to C:\Users\John\Project\prog ,then you need to

type **cd prog** and press Enter. (This new directory should contain your previously created C# file (hello.cs in our case)).

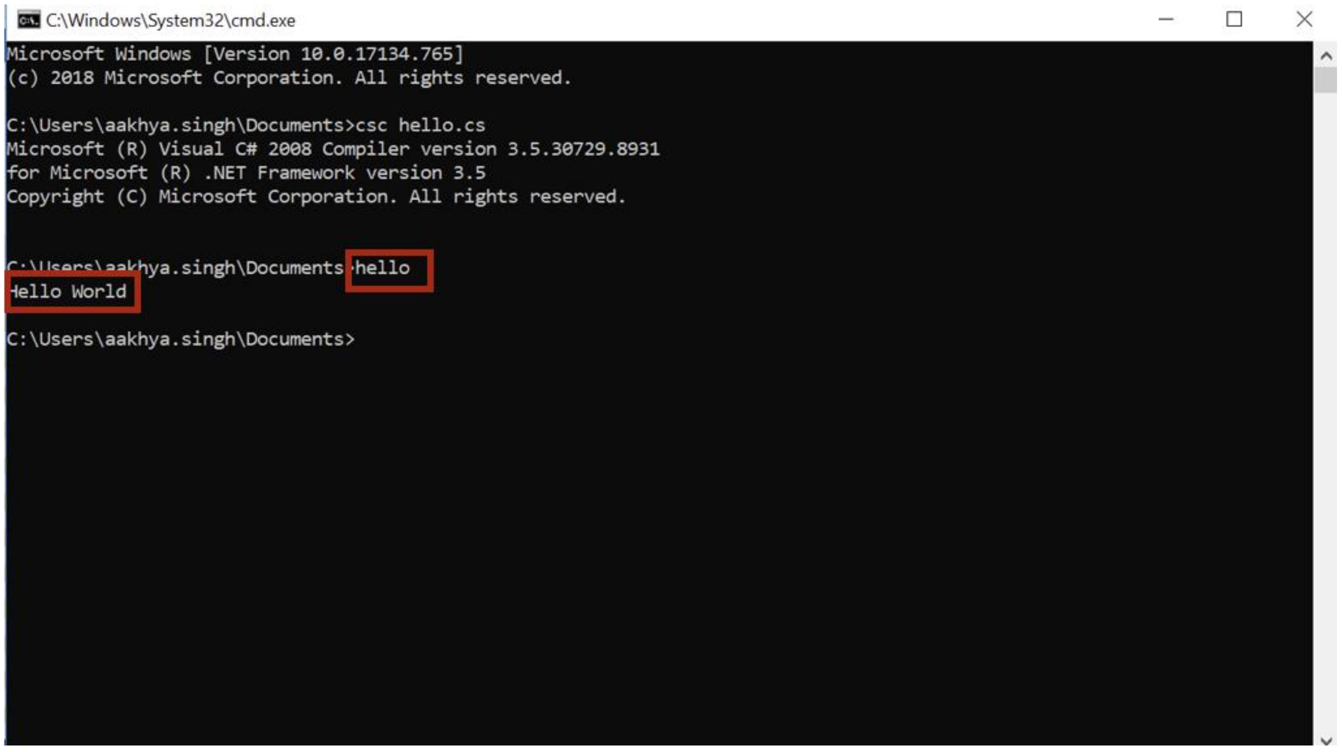
10. Once you are in the correct directory in which you have your program, then you can compile and run your program.

To compile, type `csc filename.cs` and press Enter. Here, our filename is hello.cs, so we will write `csc hello.cs`. If it shows any error, then try to remove that error from your program and then again save and compile your program.



```
Microsoft Windows [Version 10.0.17134.765]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\aakhya.singh\Documents>csc hello.cs
Microsoft (R) Visual C# 2008 Compiler version 3.5.30729.8931
for Microsoft (R) .NET Framework version 3.5
Copyright (C) Microsoft Corporation. All rights reserved.


C:\Users\aakhya.singh\Documents>_
```

11. Once the program is compiled, run the program by typing `filename` (`hello` in our case) and then pressing Enter. We will get the output as shown below.



```
Microsoft Windows [Version 10.0.17134.765]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\aakhya.singh\Documents>csc hello.cs
Microsoft (R) Visual C# 2008 Compiler version 3.5.30729.8931
for Microsoft (R) .NET Framework version 3.5
Copyright (C) Microsoft Corporation. All rights reserved.


C:\Users\aakhya.singh\Documents>hello
Hello World

C:\Users\aakhya.singh\Documents>
```

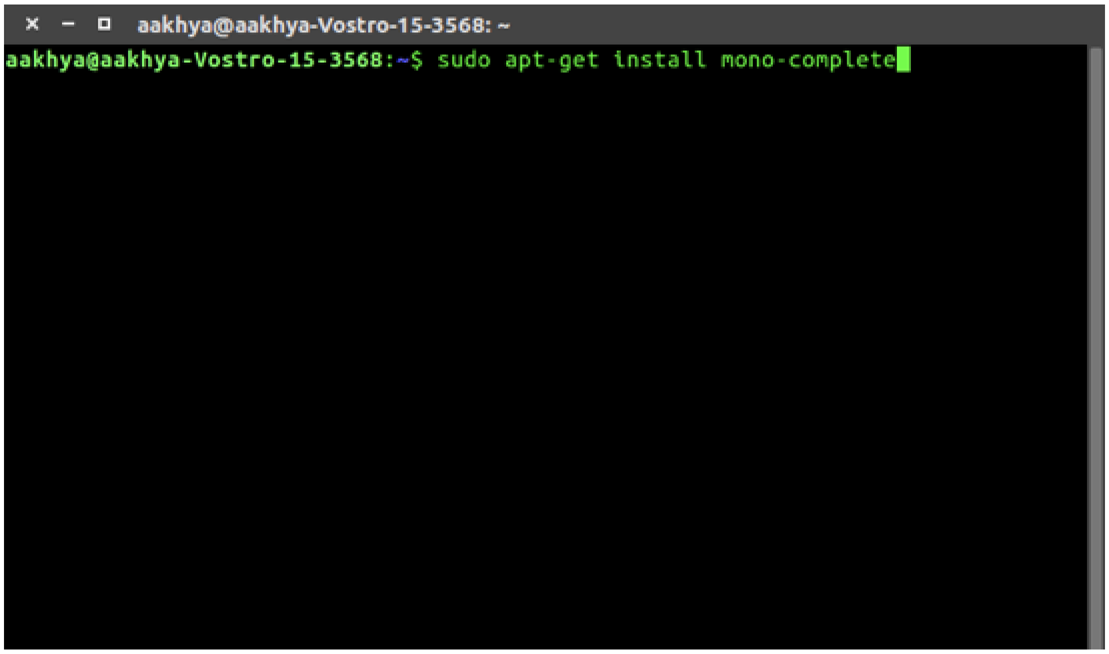
Linux

For Linux, you can write your C# program in various text editors like Vim (or vi), Sublime, Atom, etc. To compile and run our C# program in Linux, we will use **Mono** which is an open-source implementation of the .NET framework.

So let's see how to create and run a C# program on Linux.

1. Open Terminal (`ctrl+alt+T`).

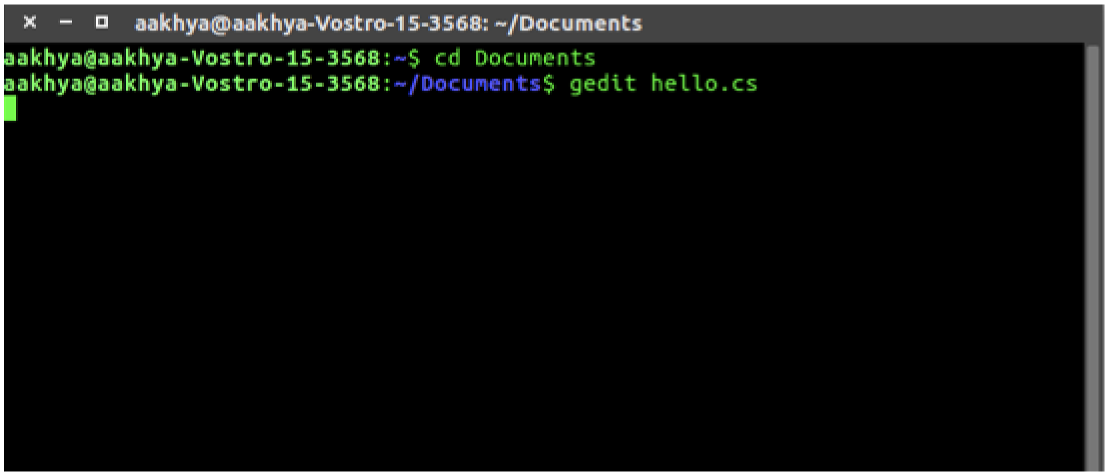
2. Type the command `sudo apt install mono-complete` to install mono-complete.



```
aakhya@aakhya-Vostro-15-3568: ~
aakhya@aakhya-Vostro-15-3568:~$ sudo apt-get install mono-complete
```

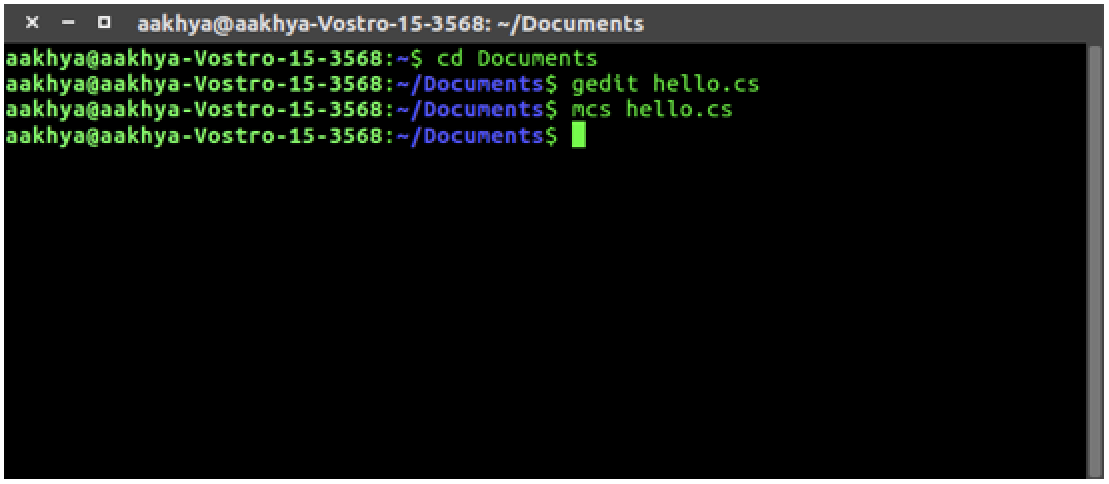
3. Open a text editor (we are going to use Gedit) and save the following program with a **.cs** extension. We are going to name our file *hello.cs*, so we will open the file using `gedit hello.cs` and write the following program and save it.

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



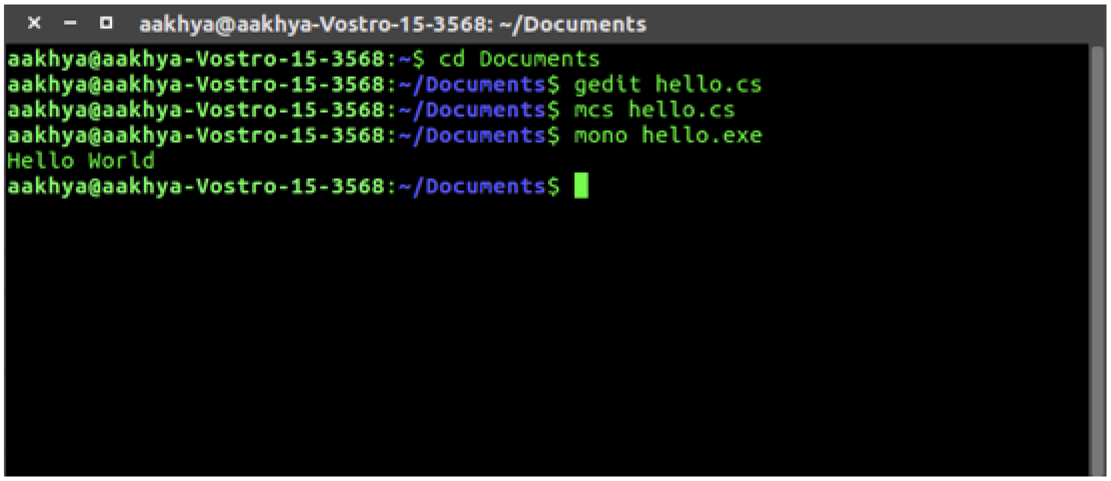
```
aakhya@aakhya-Vostro-15-3568: ~/Documents
aakhya@aakhya-Vostro-15-3568:~$ cd Documents
aakhya@aakhya-Vostro-15-3568:~/Documents$ gedit hello.cs
```

4. Now, you can compile the program using `mcs filename.cs`, so in our case, `mcs hello.cs`.



```
aakhya@aakhya-Vostro-15-3568: ~/Documents
aakhya@aakhya-Vostro-15-3568:~$ cd Documents
aakhya@aakhya-Vostro-15-3568:~/Documents$ gedit hello.cs
aakhya@aakhya-Vostro-15-3568:~/Documents$ mcs hello.cs
aakhya@aakhya-Vostro-15-3568:~/Documents$
```

5. To execute the program, write `mono filename.exe`, so in our case, `mono hello.exe`.



```
aakhya@aakhya-Vostro-15-3568: ~/Documents
aakhya@aakhya-Vostro-15-3568:~$ cd Documents
aakhya@aakhya-Vostro-15-3568:~/Documents$ gedit hello.cs
aakhya@aakhya-Vostro-15-3568:~/Documents$ mcs hello.cs
aakhya@aakhya-Vostro-15-3568:~/Documents$ mono hello.exe
Hello World
aakhya@aakhya-Vostro-15-3568:~/Documents$
```

You can see "Hello World" printed on the screen.

Mac

In Mac, you can use any text editor of your choice, we are going to use Atom. To compile and run our C# program in Mac, we will use **Mono** which is an open-source implementation of the .NET framework.

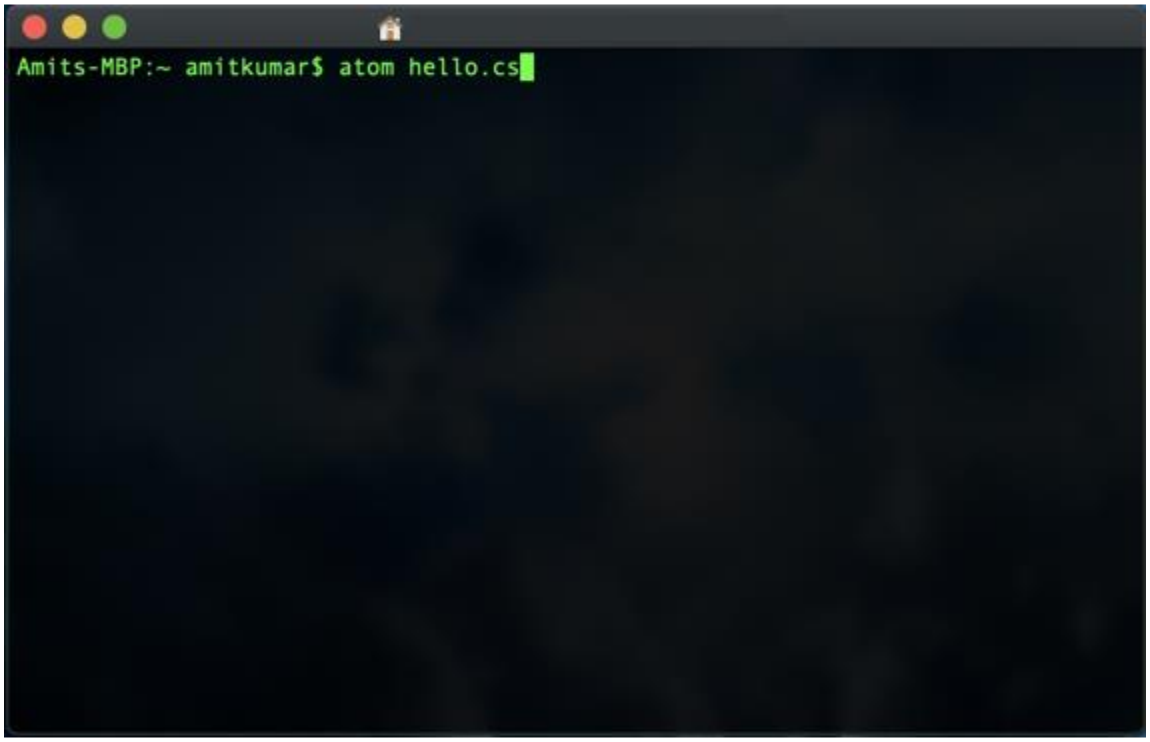
So let's see how to create and run a C# program on Mac.

1. Download and install Mono.

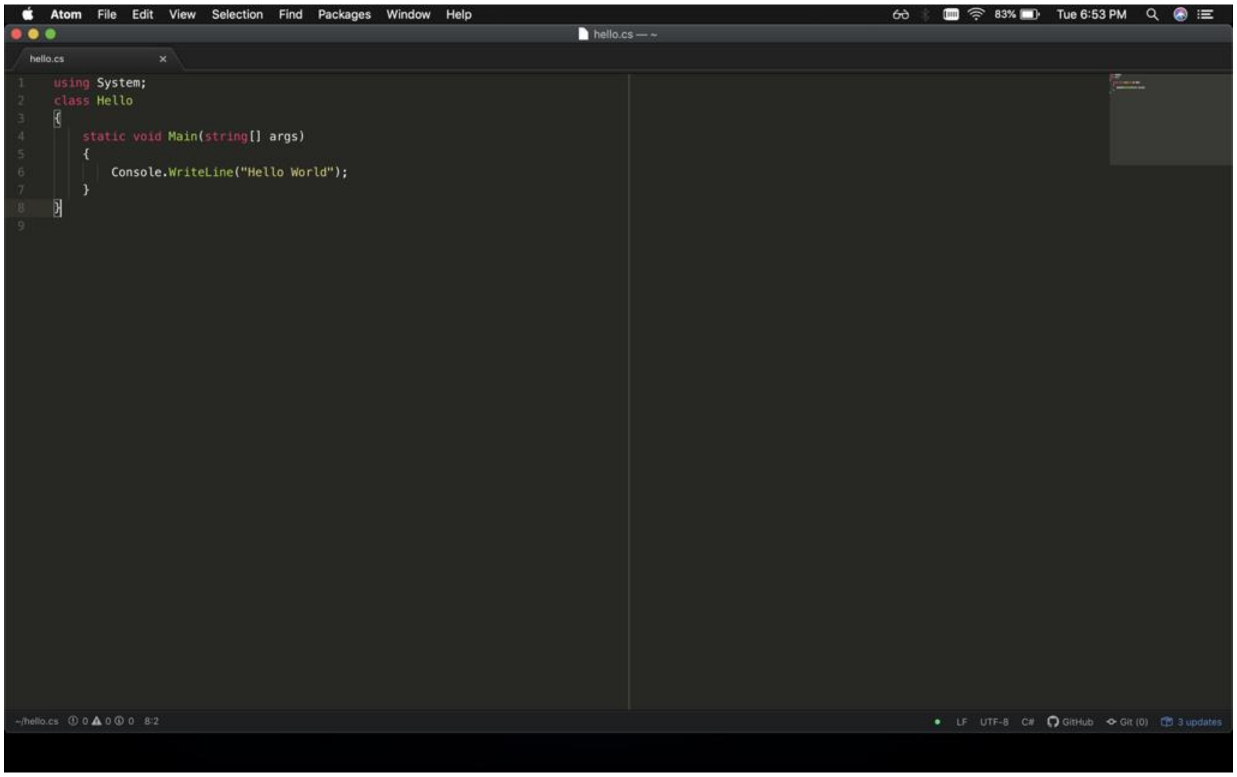
2. Open Terminal

3. Open a text editor (we are going to use Atom) and save the following program with a **.cs** extension. We are going to name our file *hello.cs*, so we will open the file using `atom hello.cs` and write the following program and save it.

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

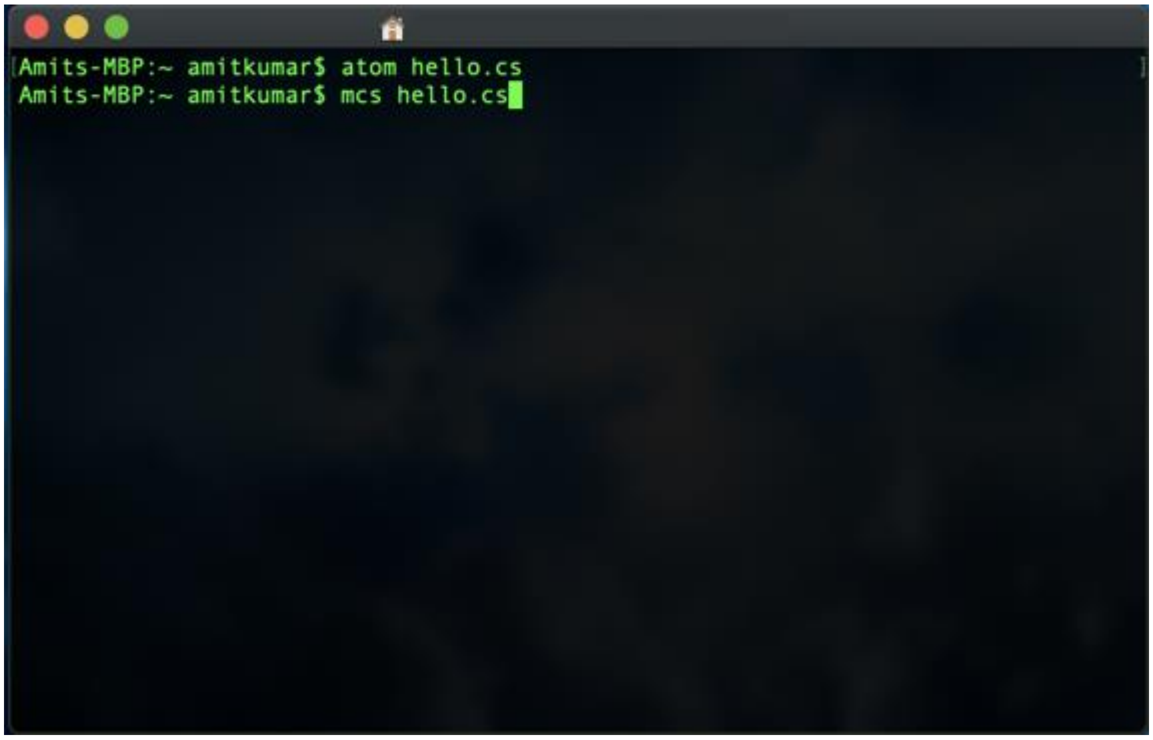


```
Amits-MBP:~ amitkumar$ atom hello.cs
```



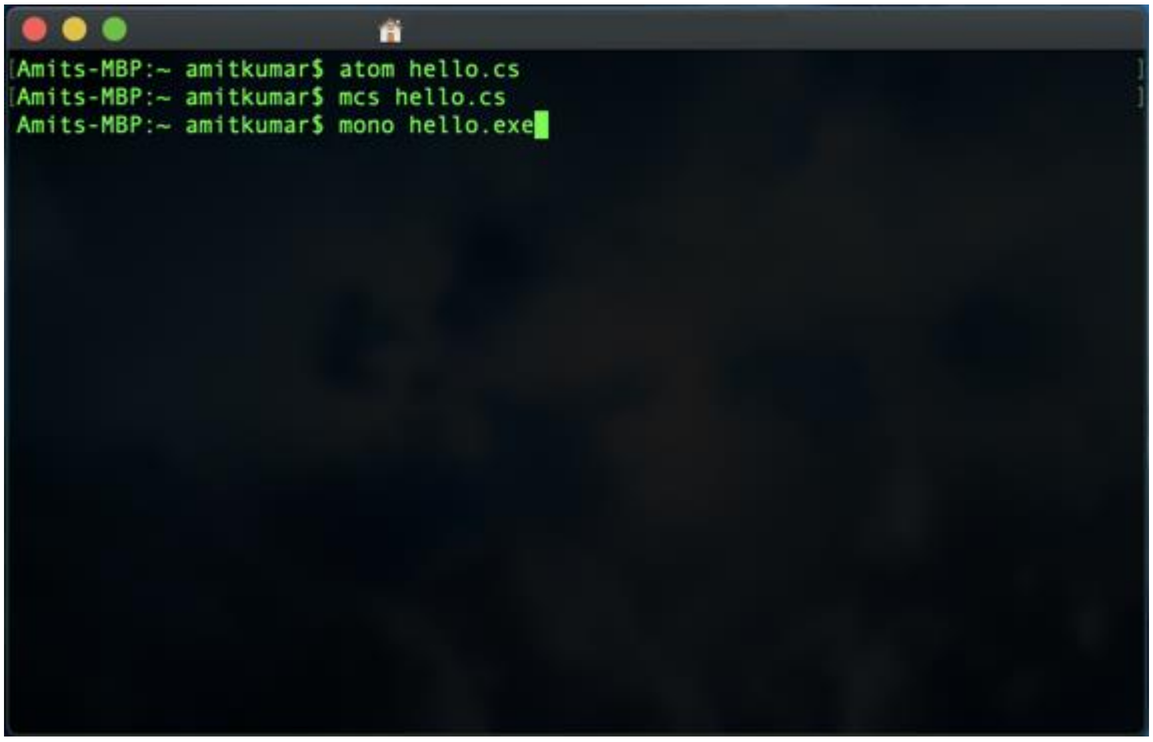
```
using System;
class Hello
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World");
    }
}
```

4. Now, you can compile the program using `mcs filename.cs`, so in our case, `mcs hello.cs`.



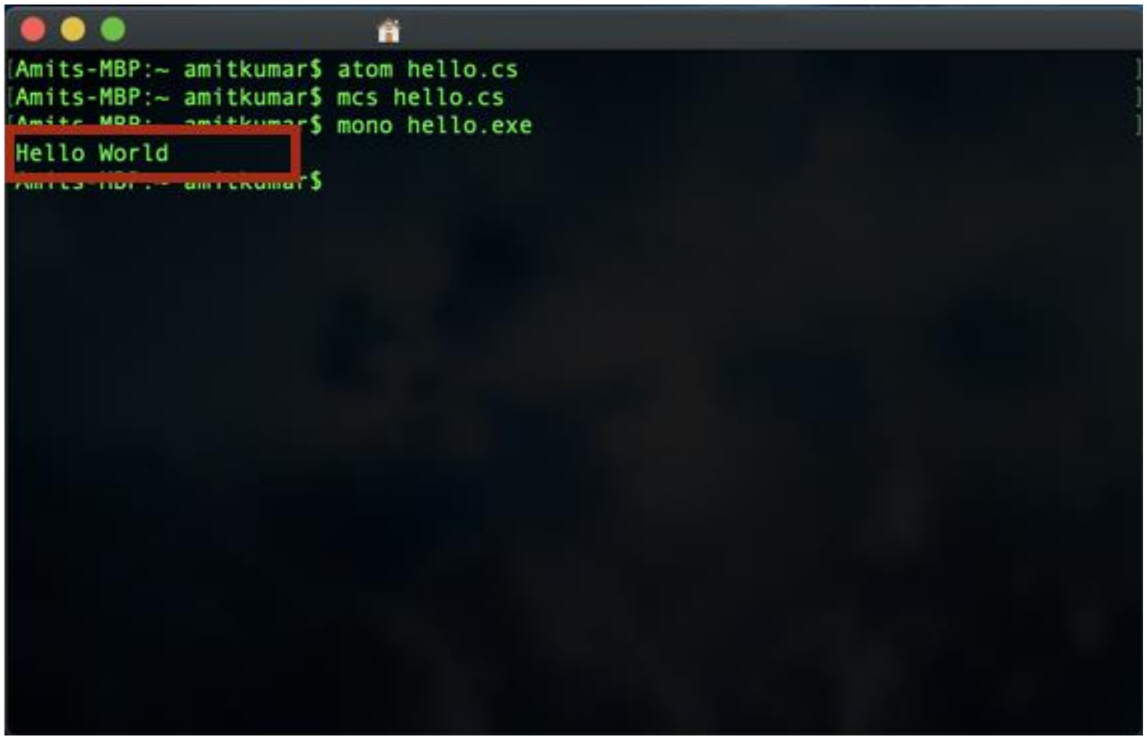
```
Amits-MBP:~ amitkumar$ atom hello.cs
Amits-MBP:~ amitkumar$ mcs hello.cs
```

5. To execute the program, write `mono filename.exe`, so in our case, `mono hello.exe`.



```
Amits-MBP:~ amitkumar$ atom hello.cs
Amits-MBP:~ amitkumar$ mcs hello.cs
Amits-MBP:~ amitkumar$ mono hello.exe
```

You can see "Hello World" printed on the screen.



```
Amits-MBP:~ amitkumar$ atom hello.cs
Amits-MBP:~ amitkumar$ mcs hello.cs
Amits-MBP:~ amitkumar$ mono hello.exe
Hello World
Amits-MBP:~ amitkumar$
```

C# Comments

The C# comments are statements that are not executed by the compiler. The comments in C# programming can be used to provide explanation of the code, variable, method or class. By the help of comments, you can hide the program code also.

There are two types of comments in C#.

- Single Line comment
- Multi Line comment

C# Single Line Comment

The single line comment starts with // (double slash). Let's see an example of single line comment in C#.

```
using System;
  public class CommentExample
   {
     public static void Main(string[] args)
      {
         int x = 10;//Here, x is a variable
         Console.WriteLine(x);
      }
```

```
    }
```

Output:

```
10
```

C# Multi Line Comment

The C# multi line comment is used to comment multiple lines of code. It is surrounded by slash and asterisk (/* ..... */). Let's see an example of multi line comment in C#.

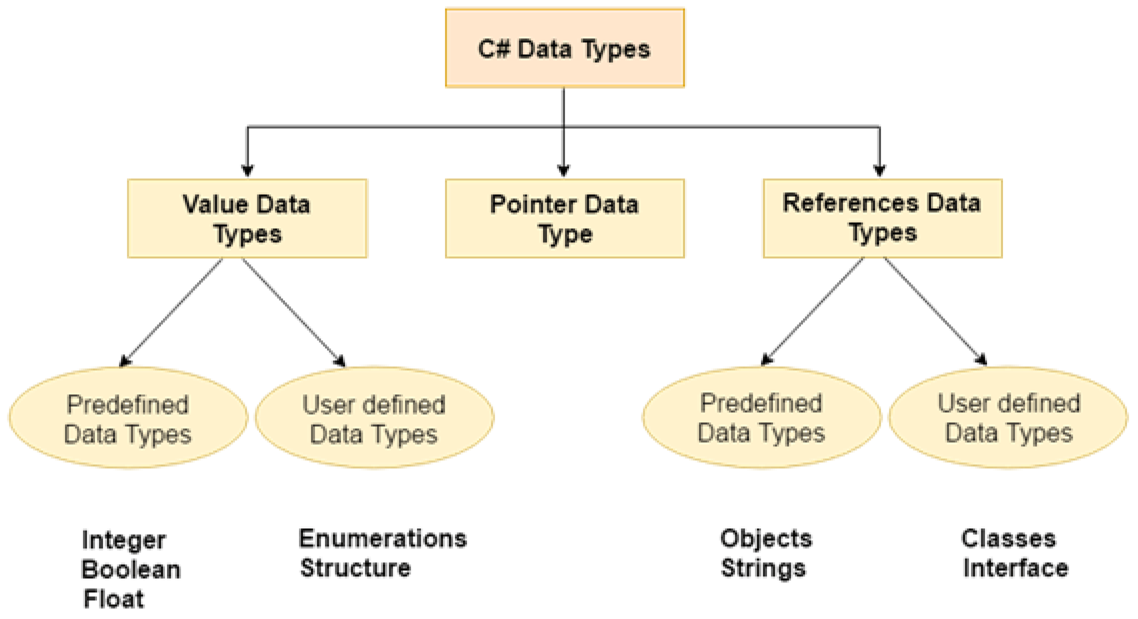
```
using System;
  public class CommentExample
   {
      public static void Main(string[] args)
       {
          /* Let's declare and
         print variable in C#. */
          int x=20;
          Console.WriteLine(x);
       }
   }
```

Output:

```
20
```

C# Data Types

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.

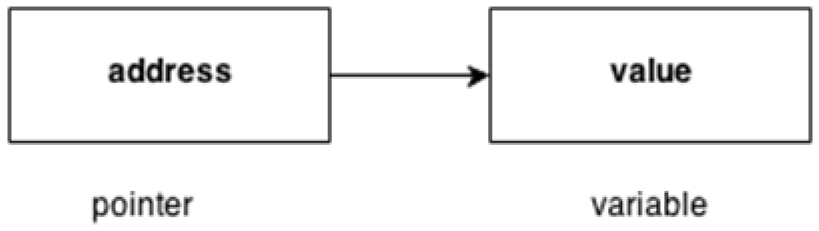
There are 2 types of reference data type in C# language.

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.



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).

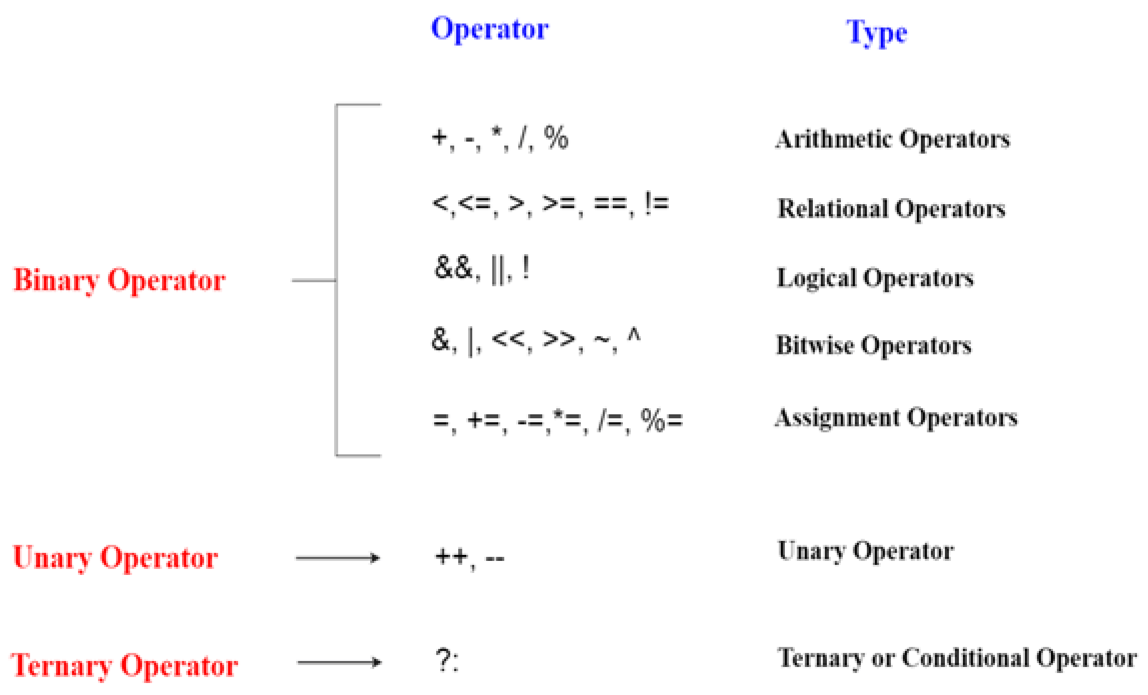
```
int * a;  //pointer to int
char * c; //pointer to char
```

C# operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc.

There are following types of operators to perform different types of operations in C# language.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Unary Operators
- Ternary Operators
- Misc Operators



Precedence of Operators in C#

The precedence of operator specifies that which operator will be evaluated first and next. The associativity specifies the operators direction to be evaluated, it may be left to right or right to left.

Let's understand the precedence by the example given below:

```
1. int data= 10+ 5*5
```

The "data" variable will contain 35 because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C# operators is given below:

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

Loops in C#

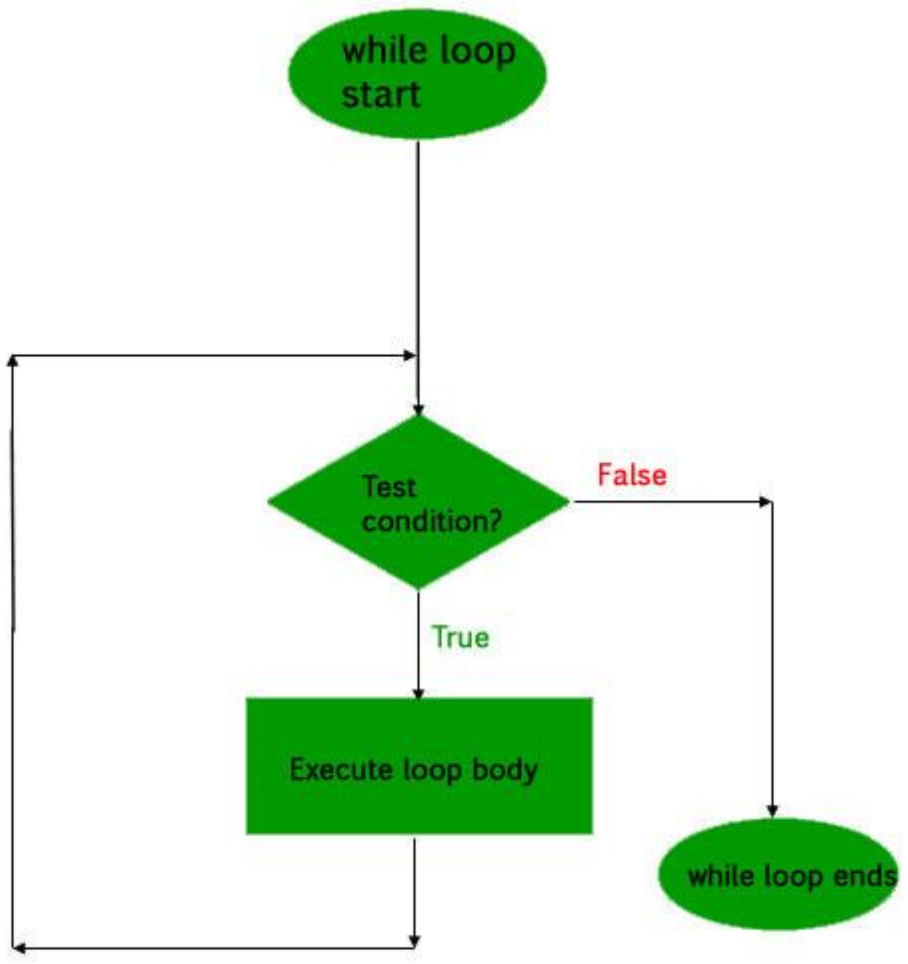
Looping in programming language is a way to execute a statement or a set of statements multiple number of times depending on the result of condition to be evaluated to execute statements. The result condition should be true to execute statements within loops.

Loops are mainly divided into two categories:
Entry Controlled Loops: The loops in which condition to be tested is present in beginning of loop body are known as **Entry Controlled Loops**. **while loop** and **for loop** are entry controlled loops.
1. while loop The test condition is given in beginning of loop and all statements are executed till the given boolean condition satisfies, when the condition becomes false, the control will be out from the while loop.

Syntax:

```
while (boolean condition)
{
   loop statements...
}
```

Flowchart:



Example:

filter_none

edit

play_arrow

brightness_4

```
// C# program to illustrate while loop
using System;


class whileLoopDemo
{
    public static void Main()
    {
        int x = 1;

        // Exit when x becomes greater than 4
        while (x <= 4)
        {
            Console.WriteLine("GeeksforGeeks");

            // Increment the value of x for
            // next iteration
            x++;
        }
    }
}
```

Output:

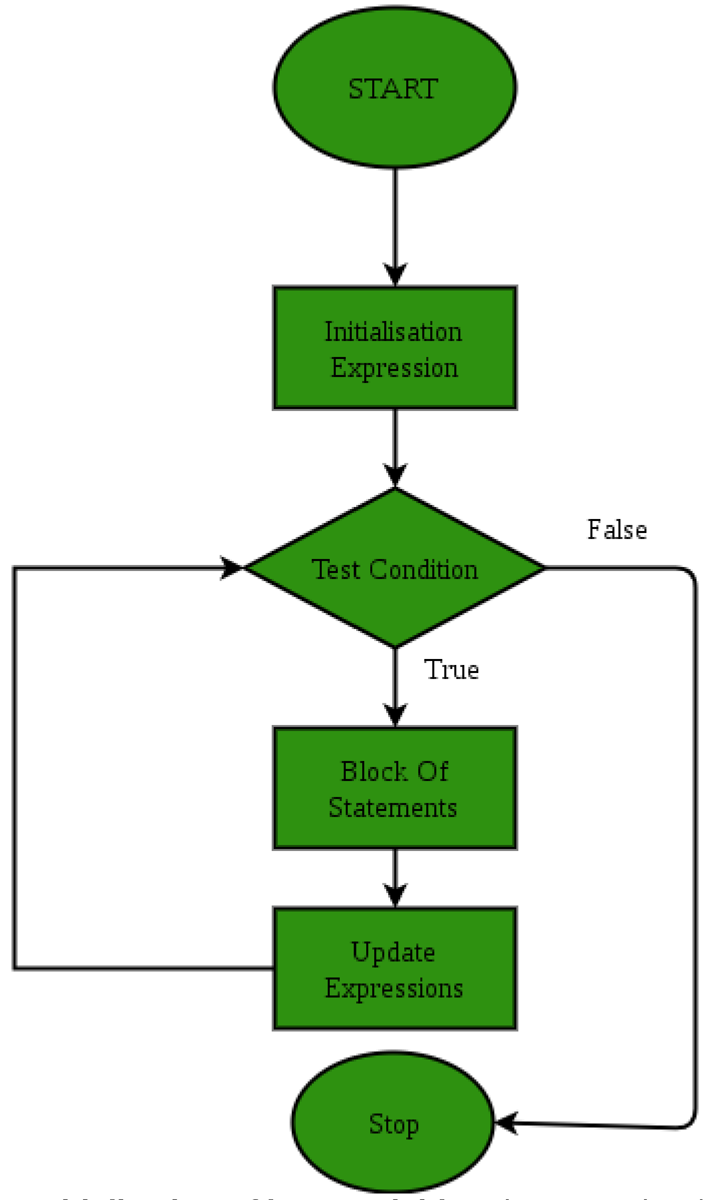
```
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
GeeksforGeeks
```

2. for loop

for loop has similar functionality as while loop but with different syntax. for loops are preferred when the number of times loop statements are to be executed is known before hand. The loop variable initialization, condition to be tested and increment / decrement of the loop variable is done in one line in for loop thereby providing a shorter, easy to debug structure of looping.

```
for (loop variable initialization ; testing condition;
                              increment / decrement)
{
    // statements to be executed
}
```

Flowchart:



1. Initialization of loop variable: Th expression / variable controlling the loop is initialized here. It is the starting point of for loop. An already declared variable can be used or a variable can be declared, local to loop only.
2. Testing Condition: The testing condition to execute statements of loop. It is used for testing the exit condition for a loop. It must return a boolean value **true or false**. When the condition became false the control will be out from the loop and for loop ends.
3. Increment / Decrement: The loop variable is incremented / decremented according to the requirement and the control then shifts to the testing condition again.
Note: Initialization part is evaluated only once when the for loop starts.
Example:

filter_none

edit

play_arrow
brightness_4

```
// C# program to illustrate for loop.

using System;



class forLoopDemo

{

    public static void Main()

    {

        // for loop begins when x=1

        // and runs till x <=4

        for (int x = 1; x <= 4; x++)

            Console.WriteLine("GeeksforGeeks");

    }

}
```

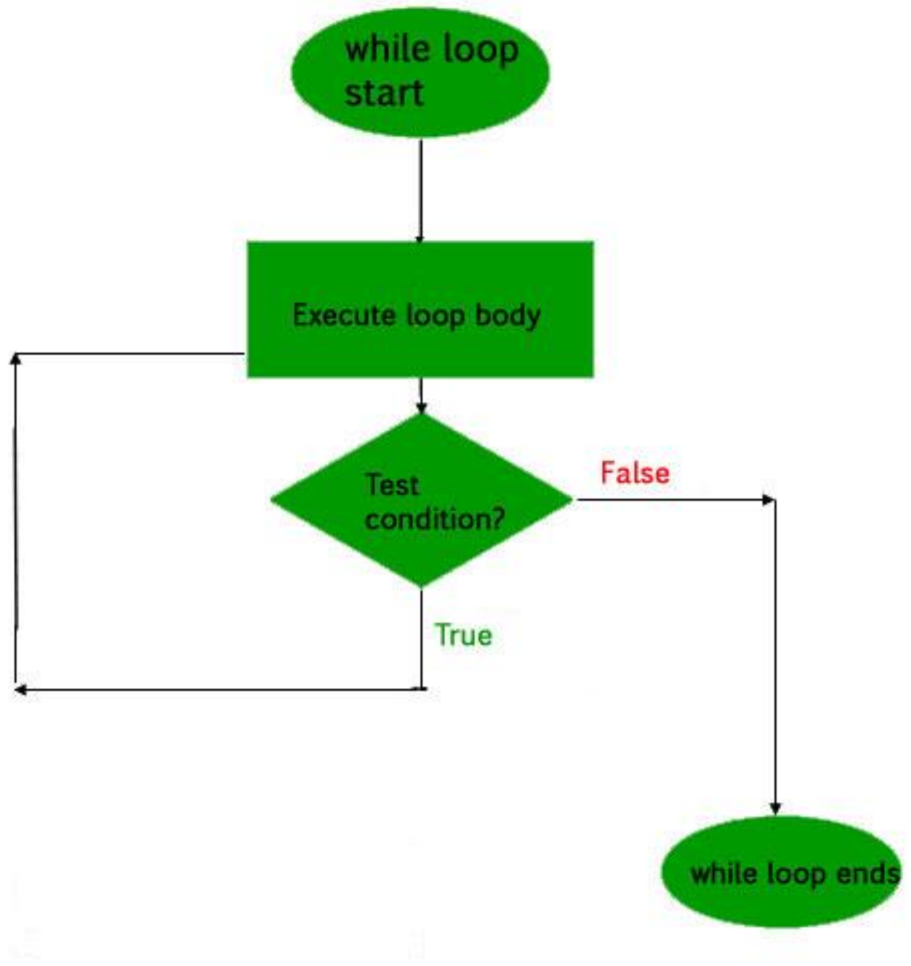
Output:

```
GeeksforGeeks

GeeksforGeeks

GeeksforGeeks

GeeksforGeeks
```

Exit Controlled Loops: The loops in which the testing condition is present at the end of loop body are termed as **Exit Controlled Loops**. **do-while** is an exit controlled loop.
Note: In Exit Controlled Loops, loop body will be evaluated for at-least one time as the testing condition is present at the end of loop body.
1. do-while loop
do while loop is similar to while loop with only difference that it checks the condition after executing the statements, i.e it will execute the loop body one time for sure because it checks the condition after executing the statements.
Syntax :

```
do

{

    statements..
}while (condition);
```

Flowchart:



Example:

filter_none

edit

play_arrow

brightness_4

```
// C# program to illustrate do-while loop
using System;


class dowhileloopDemo
{
    public static void Main()
    {
        int x = 21;
        do
        {
            // The line will be printed even
            // if the condition is false
            Console.WriteLine("GeeksforGeeks");
            x++;
        }
        while (x < 20);
    }
}
```

Output:

```
GeeksforGeeks
```

Infinite Loops:
The loops in which the test condition does not evaluate false ever are tend to execute statements forever until an external force is used to end it and thus they are known as infinite loops.
Example:

filter_none

edit
play_arrow
brightness_4

```
// C# program to demonstrate infinite loop
using System;


class infiniteLoop
{
    public static void Main()
    {
        // The statement will be printed
        // infinite times
        for(;;)
        Console.WriteLine("This is printed infinite times");
    }
}
```

Output:

```
This is printed infinite times
This is printed infinite times
This is printed infinite times
This is printed infinite times
This is printed infinite times
This is printed infinite times
This is printed infinite times
..........
```

Nested Loops:
When loops are present inside the other loops, it is known as nested loops.

Example:

filter_none

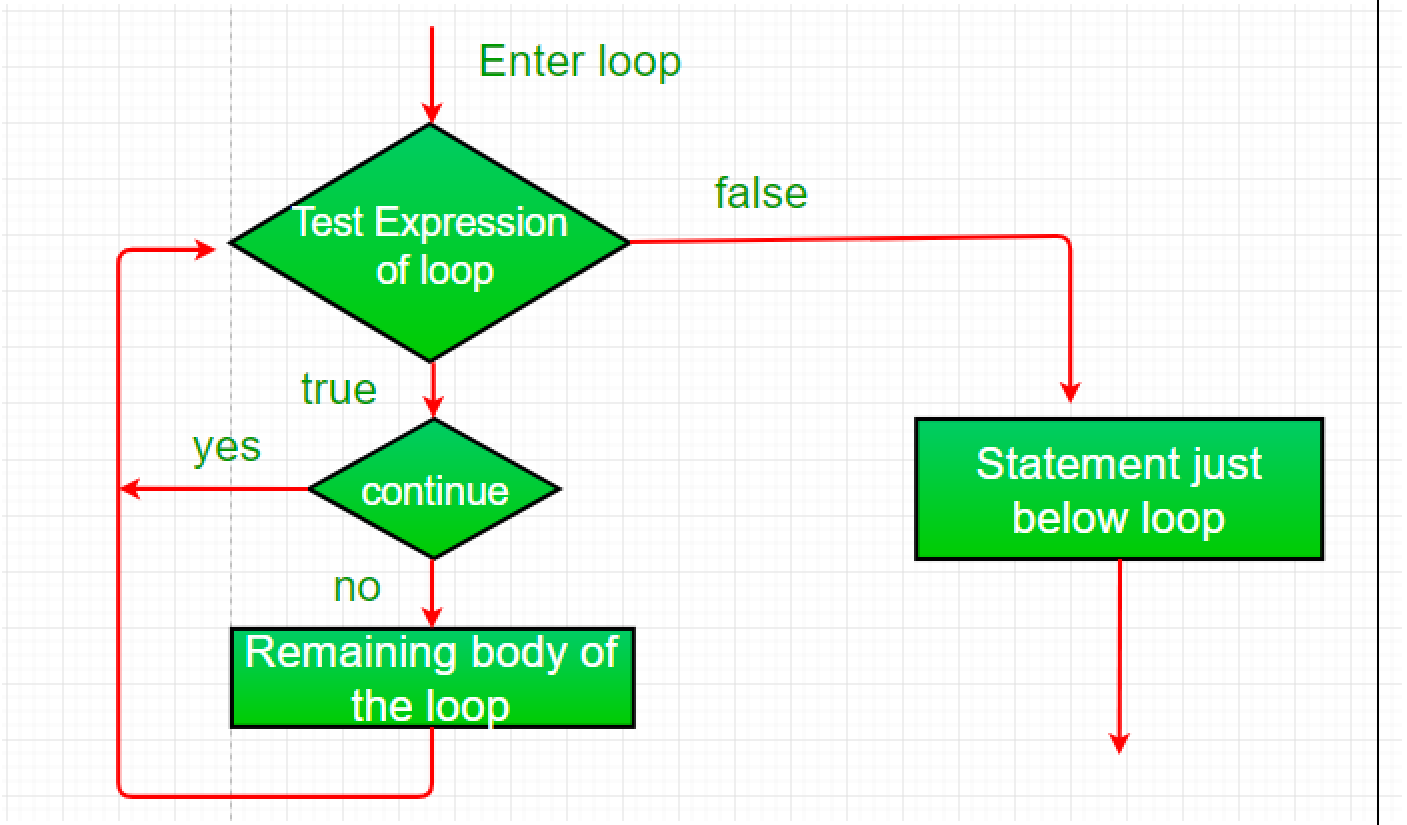
edit
play_arrow
brightness_4

```
// C# program to demonstrate nested loops
using System;

class nestedLoops
{
    public static void Main()
    {
        // loop within loop printing GeeksforGeeks
       for(int i = 2; i < 3; i++)
             for(int j = 1; j < i; j++)
                 Console.WriteLine("GeeksforGeeks");
    }
}
```

Output:

```
GeeksforGeeks
```

continue statement:
continue statement is used to skip over the execution part of loop on a certain condition and move the flow to next updation part.
Flowchart:



Example:
filter_none

edit
play_arrow
brightness_4

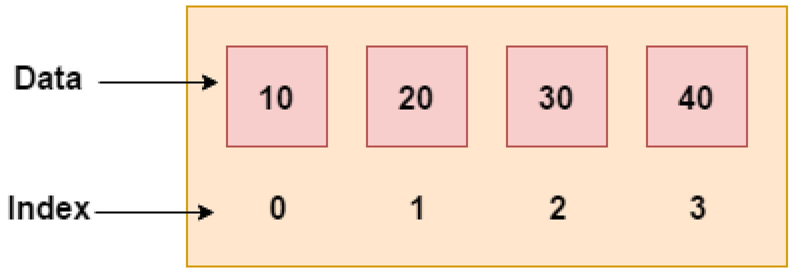
```
// C# program to demonstrate continue statement
using System;


class demoContinue
{
    public static void Main()
    {
        // GeeksforGeeks is printed only 2 times
        // because of continue statement
        for(int i = 1; i < 3; i++)
        {
            if(i == 2)
              continue;

            Console.WriteLine("GeeksforGeeks");
        }
    }
}
```

Output:

```
GeeksforGeeks
```

C# Arrays

Like other programming languages, array in C# is a group of similar types of elements that have contiguous memory location. In C#, array is an *object* of base type **System.Array**. In C#, array index starts from 0. We can store only fixed set of elements in C# array.



Advantages of C# Array

- Code Optimization (less code)
- Random Access
- Easy to traverse data
- Easy to manipulate data
- Easy to sort data etc.

Disadvantages of C# Array

- Fixed size

C# Array Types

There are 3 types of arrays in C# programming:

1. Single Dimensional Array
2. Multidimensional Array
3. Jagged Array

C# Single Dimensional Array

To create single dimensional array, you need to use square brackets [] after the type.

```
int[] arr = new int[5];//creating array
```

You cannot place square brackets after the identifier.

```
int arr[] = new int[5];//compile time error
```

Let's see a simple example of C# array, where we are going to declare, initialize and traverse array.

```
using System;
public class ArrayExample
{
    public static void Main(string[] args)
    {
        int[] arr = new int[5];//creating array
        arr[0] = 10;//initializing array
        arr[2] = 20;
        arr[4] = 30;

        //traversing array
        for (int i = 0; i < arr.Length; i++)
        {
            Console.WriteLine(arr[i]);
        }
    }
}
```

Output:

```
10
0
20
0
30
```

C# Array Example: Declaration and Initialization at same time

There are 3 ways to initialize array at the time of declaration.

```csharp
int[] arr = new int[5]{ 10, 20, 30, 40, 50 };
```

We can omit the size of array.

```csharp
int[] arr = new int[]{ 10, 20, 30, 40, 50 };
```

We can omit the new operator also.

```csharp
int[] arr = { 10, 20, 30, 40, 50 };
```

Let's see the example of array where we are declaring and initializing array at the same time.

```csharp
using System;
public class ArrayExample
{
    public static void Main(string[] args)
    {
        int[] arr = { 10, 20, 30, 40, 50 };//Declaration and Initialization of array

        //traversing array
        for (int i = 0; i < arr.Length; i++)
        {
            Console.WriteLine(arr[i]);
        }
    }
}
```

Output:

```
10
20
30
40
50
```

C# Array Example: Traversal using foreach loop

We can also traverse the array elements using foreach loop. It returns array element one by one.

```
using System;
public class ArrayExample
{
    public static void Main(string[] args)
    {
        int[] arr = { 10, 20, 30, 40, 50 };//creating and initializing array

        //traversing array
        foreach (int i in arr)
        {
            Console.WriteLine(i);
        }
    }
}
```

Output:

```
10
20
30
40
50
```

C# Command Line Arguments

Arguments that are passed by command line known as command line arguments. We can send arguments to the Main method while executing the code. The string **args** variable contains all the values passed from the command line.

In the following example, we are passing command line arguments during execution of program.

C# Command Line Arguments Example

```
using System;
namespace CSharpProgram
{
    class Program
    {
        // Main function, execution entry point of the program
        static void Main(string[] args) // string type parameters
        {
            // Command line arguments
            Console.WriteLine("Argument length: "+args.Length);
            Console.WriteLine("Supplied Arguments are:");
            foreach (Object obj in args)
            {
                Console.WriteLine(obj);
            }
        }
    }
}
```

Compile and execute this program by using following commands.

Compile: csc Program.cs

Execute: Program.exe Hi there, how are you?

After executing the code, it produces the following output to the console.

Output:

```
Argument length: 5
Supplied Arguments are:
Hi
there,
how
are
you?
```

C# Function

Function is a block of code that has a signature. Function is used to execute statements specified in the code block. A function consists of the following components:

Function name: It is a unique name that is used to make Function call.

Return type: It is used to specify the data type of function return value.

Body: It is a block that contains executable statements.

Access specifier: It is used to specify function accessibility in the application.

Parameters: It is a list of arguments that we can pass to the function during call.

C# Function Syntax

```
<access-specifier><return-type>FunctionName(<parameters>)
{
// function body
// return statement
}
```

Access-specifier, parameters and return statement are optional.

Let's see an example in which we have created a function that returns a string value and takes a string parameter.

C# Function: using no parameter and return type

A function that does not return any value specifies **void** type as a return type. In the following example, a function is created without return type.

```
using System;
namespace FunctionExample
{
    class Program
    {
        // User defined function without return type
        public void Show() // No Parameter
        {
            Console.WriteLine("This is non parameterized function");
            // No return statement
        }
        // Main function, execution entry point of the program
        static void Main(string[] args)
        {
            Program program = new Program(); // Creating Object
            program.Show(); // Calling Function
        }
    }
}
```

Output:

```
This is non parameterized function
```

C# Function: using parameter but no return type

```
using System;
```

```
namespace FunctionExample
{
    class Program
    {
        // User defined function without return type
        public void Show(string message)
        {
            Console.WriteLine("Hello " + message);
            // No return statement
        }
       // Main function, execution entry point of the program
        static void Main(string[] args)
        {
            Program program = new Program(); // Creating Object
            program.Show("Rahul Kumar"); // Calling Function
        }
    }
}
```

Output:

```
Hello Rahul Kumar
```

A function can have zero or any number of parameters to get data. In the following example, a function is created without parameters. A function without parameter is also known as **non-parameterized** function.

C# Function: using parameter and return type

```
using System;
namespace FunctionExample
{
    class Program
    {
        // User defined function
        public string Show(string message)
        {
         Console.WriteLine("Inside Show Function");
         return message;
        }
        // Main function, execution entry point of the program
        static void Main(string[] args)
        {
            Program program = new Program();
            string message = program.Show("Rahul Kumar");
            Console.WriteLine("Hello "+message);
        }
    }
}
```

Output:

```
Inside Show Function
Hello Rahul Kumar
```

C# Call By Value

In C#, value-type parameters are that pass a copy of original value to the function rather than reference. It does not modify the original value. A change made in passed value does not alter the actual value. In the following example, we have pass value during function call.

C# Call By Value Example

```
using System;
namespace CallByValue
{
    class Program
    {
        // User defined function
        public void Show(int val)
        {
             val *= val; // Manipulating value
            Console.WriteLine("Value inside the show function "+val);
            // No return statement
        }
        // Main function, execution entry point of the program
        static void Main(string[] args)
        {
            int val = 50;
            Program program = new Program(); // Creating Object
            Console.WriteLine("Value before calling the function "+val);
            program.Show(val); // Calling Function by passing value
            Console.WriteLine("Value after calling the function " + val);
        }
    }
}
```

Output:

```
Value before calling the function 50
Value inside the show function 2500
Value after calling the function 50
```

C# Call By Reference

C# provides a **ref** keyword to pass argument as reference-type. It passes reference of arguments to the function rather than copy of original value. The changes in passed values are permanent and **modify** the original variable value.

C# Call By Reference Example

```csharp
using System;
namespace CallByReference
{
    class Program
    {
        // User defined function
        public void Show(ref int val)
        {
             val *= val; // Manipulating value
            Console.WriteLine("Value inside the show function "+val);
            // No return statement
        }
        // Main function, execution entry point of the program
        static void Main(string[] args)
        {
            int val = 50;
            Program program = new Program(); // Creating Object
            Console.WriteLine("Value before calling the function "+val);
            program.Show(ref val); // Calling Function by passing reference
            Console.WriteLine("Value after calling the function " + val);
        }
    }
}
```

Output:

```
Value before calling the function 50
Value inside the show function 2500
Value after calling the function 2500
```

C# Out Parameter

C# provides **out** keyword to pass arguments as out-type. It is like reference-type, except that it does not require variable to initialize before passing. We must use **out** keyword to pass argument as out-type. It is useful when we want a function to return multiple values.

C# Out Parameter Example 1

```csharp
using System;
namespace OutParameter
{
    class Program
    {
        // User defined function
        public void Show(out int val) // Out parameter
        {
            int square = 5;
            val = square;
            val *= val; // Manipulating value
        }
        // Main function, execution entry point of the program
        static void Main(string[] args)
        {
            int val = 50;
            Program program = new Program(); // Creating Object
            Console.WriteLine("Value before passing out variable " + val);
            program.Show(out val); // Passing out argument
            Console.WriteLine("Value after recieving the out variable " + val);
        }
    }
}
```

Output:

```
Value before passing out variable 50
Value after receiving the out variable 25
```

The following example demonstrates that how a function can return multiple values.

C# Out Parameter Example 2

```
using System;
namespace OutParameter
{
    class Program
    {
        // User defined function
        public void Show(out int a, out int b) // Out parameter
        {
            int square = 5;
            a = square;
            b = square;
            // Manipulating value
            a *= a;
            b *= b;
        }
        // Main function, execution entry point of the program
        static void Main(string[] args)
        {
            int val1 = 50, val2 = 100;
            Program program = new Program(); // Creating Object
            Console.WriteLine("Value before passing \n val1 = " + val1+" \n val2 = "+val2);
            program.Show(out val1, out val2); // Passing out argument
            Console.WriteLine("Value after passing \n val1 = " + val1 + " \n val2 = " + val2);
        }
    }
}
```

Output:

```
Value before passing
 val1 = 50
 val2 = 100
Value after passing
 val1 = 25
 val2 = 25
```

C# Object and Class

Since C# is an object-oriented language, program is designed using objects and classes in C#.

C# Object

In C#, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.

In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.

Object is a runtime entity, it is created at runtime.

Object is an instance of a class. All the members of the class can be accessed through object.

Let's see an example to create object using new keyword.

```
Student s1 = new Student();//creating an object of Student
```

In this example, Student is the type and s1 is the reference variable that refers to the instance of Student class. The new keyword allocates memory at runtime.

C# Class

In C#, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Let's see an example of C# class that has two fields only.

```
public class Student
 {
     int id;//field or data member
     String name;//field or data member
 }
```

C# Object and Class Example

Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the object and prints the object value.

```
using System;
   public class Student
    {
        int id;//data member (also instance variable)
        String name;//data member(also instance variable)

    public static void Main(string[] args)
       {
           Student s1 = new Student();//creating an object of Student
           s1.id = 101;
           s1.name = "Sonoo Jaiswal";
           Console.WriteLine(s1.id);
           Console.WriteLine(s1.name);

       }
   }
```

Output:

```
101
Sonoo Jaiswal
```

C# Class Example 2: Having Main() in another class

Let's see another example of class where we are having Main() method in another class. In such case, class must be public.

```
using System;
   public class Student
    {
        public int id;
        public String name;
   }
   class TestStudent{
       public static void Main(string[] args)
       {
           Student s1 = new Student();
           s1.id = 101;
           s1.name = "Sonoo Jaiswal";
           Console.WriteLine(s1.id);
           Console.WriteLine(s1.name);

       }
   }
```

Output:

```
101
Sonoo Jaiswal
```

C# Class Example 3: Initialize and Display data through method

Let's see another example of C# class where we are initializing and displaying object through method.

```csharp
using System;
  public class Student
   {
       public int id;
       public String name;
       public void insert(int i, String n)
       {
           id = i;
           name = n;
       }
       public void display()
       {
           Console.WriteLine(id + " " + name);
       }
  }
  class TestStudent{
      public static void Main(string[] args)
       {
           Student s1 = new Student();
           Student s2 = new Student();
           s1.insert(101, "Ajeet");
           s2.insert(102, "Tom");
           s1.display();
           s2.display();

       }
   }
```

Output:

```
101 Ajeet
102 Tom
```

C# Class Example 4: Store and Display Employee Information

```csharp
using System;
  public class Employee
  {
      public int id;
      public String name;
      public float salary;
      public void insert(int i, String n,float s)
      {
          id = i;
          name = n;
          salary = s;
      }
      public void display()
      {
          Console.WriteLine(id + " " + name+" "+salary);
      }
 }
 class TestEmployee{
     public static void Main(string[] args)
      {
          Employee e1 = new Employee();
          Employee e2 = new Employee();
          e1.insert(101, "Sonoo",890000f);
          e2.insert(102, "Mahesh", 490000f);
          e1.display();
          e2.display();

      }
  }
```

Output:

```
101 Sonoo 890000
102 Mahesh 490000
```

C# Constructor

In C#, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C# has the same name as class or struct.

There can be two types of constructors in C#.

- Default constructor

- Parameterized constructor

C# Default Constructor

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

C# Default Constructor Example: Having Main() within class

```
using System;
  public class Employee
   {
      public Employee()
      {
         Console.WriteLine("Default Constructor Invoked");
      }
      public static void Main(string[] args)
      {
         Employee e1 = new Employee();
         Employee e2 = new Employee();
      }
   }
```

Output:

```
Default Constructor Invoked
Default Constructor Invoked
```

C# Default Constructor Example: Having Main() in another class

Let's see another example of default constructor where we are having Main() method in another class.

```
using System;
  public class Employee
   {
      public Employee()
      {
         Console.WriteLine("Default Constructor Invoked");
      }
   }
  class TestEmployee{
     public static void Main(string[] args)
```

```
    {
        Employee e1 = new Employee();
        Employee e2 = new Employee();
    }
 }
```

Output:

```
Default Constructor Invoked
Default Constructor Invoked
```

C# Parameterized Constructor

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

```
using System;
   public class Employee
    {
        public int id;
        public String name;
        public float salary;
        public Employee(int i, String n,float s)
        {
            id = i;
            name = n;
            salary = s;
        }
        public void display()
        {
            Console.WriteLine(id + " " + name+" "+salary);
        }
   }
   class TestEmployee{
       public static void Main(string[] args)
        {
            Employee e1 = new Employee(101, "Sonoo", 890000f);
            Employee e2 = new Employee(102, "Mahesh", 490000f);
            e1.display();
            e2.display();

        }
    }
```

Output:

```
101 Sonoo 890000
102 Mahesh 490000
```

C# Destructor

A destructor works opposite to constructor, It destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

Note: C# destructor cannot have parameters. Moreover, modifiers can't be applied on destructors.

C# Constructor and Destructor Example

Let's see an example of constructor and destructor in C# which is called automatically.

```
using System;
   public class Employee
    {
        public Employee()
        {
            Console.WriteLine("Constructor Invoked");
        }
        ~Employee()
        {
            Console.WriteLine("Destructor Invoked");
        }
    }
   class TestEmployee{
       public static void Main(string[] args)
        {
            Employee e1 = new Employee();
            Employee e2 = new Employee();
        }
    }
```

Output:

```
Constructor Invoked
Constructor Invoked
```

```
Destructor Invoked
Destructor Invoked
```

Note: Destructor can't be public. We can't apply any modifier on destructors.

C# this

In c# programming, this is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword in C#.

- It can be used **to refer current class instance variable**. It is used if field names (instance variables) and parameter names are same, that is why both can be distinguish easily.
- It can be used **to pass current object as a parameter to another method**.
- It can be used **to declare indexers**.

C# this example

Let's see the example of this keyword in C# that refers to the fields of current class.

```
using System;
  public class Employee
   {
       public int id;
       public String name;
       public float salary;
       public Employee(int id, String name,float salary)
       {
           this.id = id;
           this.name = name;
           this.salary = salary;
       }
       public void display()
       {
           Console.WriteLine(id + " " + name+" "+salary);
       }
  }
  class TestEmployee{
      public static void Main(string[] args)
```

```
    {
        Employee e1 = new Employee(101, "Sonoo", 890000f);
        Employee e2 = new Employee(102, "Mahesh", 490000f);
        e1.display();
        e2.display();

    }
}
```

Output:

```
101 Sonoo 890000
102 Mahesh 490000
```

C# static

In C#, static is a keyword or modifier that belongs to the type not instance. So instance is not required to access the static members. In C#, static can be field, method, constructor, class, properties, operator and event.

Note: Indexers and destructors cannot be static.

Advantage of C# static keyword

Memory efficient: Now we don't need to create instance for accessing the static members, so it saves memory. Moreover, it belongs to the type, so it will not get memory each time when instance is created.

C# Static Field

A field which is declared as static, is called static field. Unlike instance field which gets memory each time whenever you create object, there is only one copy of static field created in the memory. It is shared to all the objects.

It is used to refer the common property of all objects such as rateOfInterest in case of Account, companyName in case of Employee etc.

C# static field example

Let's see the simple example of static field in C#.

```
using System;
```

```
  public class Account
   {
       public int accno;
       public String name;
       public static float rateOfInterest=8.8f;
       public Account(int accno, String name)
       {
           this.accno = accno;
           this.name = name;
       }

       public void display()
       {
           Console.WriteLine(accno + " " + name + " " + rateOfInterest);
       }
  }
  class TestAccount{
      public static void Main(string[] args)
       {
        Account a1 = new Account(101, "Sonoo");
           Account a2 = new Account(102, "Mahesh");
           a1.display();
           a2.display();

       }
   }
```

Output:

```
101 Sonoo 8.8
102 Mahesh 8.8
```

C# static field example 2: changing static field

If you change the value of static field, it will be applied to all the objects.

```
using System;
  public class Account
   {
       public int accno;
       public String name;
       public static float rateOfInterest=8.8f;
       public Account(int accno, String name)
```

```
        {
            this.accno = accno;
            this.name = name;
        }

        public void display()
        {
            Console.WriteLine(accno + " " + name + " " + rateOfInterest);
        }
    }
    class TestAccount{
        public static void Main(string[] args)
        {
            Account.rateOfInterest = 10.5f;//changing value
            Account a1 = new Account(101, "Sonoo");
            Account a2 = new Account(102, "Mahesh");
            a1.display();
            a2.display();

        }
    }
```

Output:

```
101 Sonoo 10.5
102 Mahesh 10.5
```

C# static field example 3: Counting Objects

Let's see another example of static keyword in C# which counts the objects.

```
using System;
   public class Account
    {
        public int accno;
        public String name;
        public static int count=0;
        public Account(int accno, String name)
        {
            this.accno = accno;
            this.name = name;
            count++;
        }

        public void display()
        {
            Console.WriteLine(accno + " " + name);
        }
   }
   class TestAccount{
      public static void Main(string[] args)
        {
            Account a1 = new Account(101, "Sonoo");
            Account a2 = new Account(102, "Mahesh");
            Account a3 = new Account(103, "Ajeet");
            a1.display();
            a2.display();
            a3.display();
            Console.WriteLine("Total Objects are: "+Account.count);
        }
    }
```

Output:

```
101 Sonoo
102 Mahesh
103 Ajeet
Total Objects are: 3
```

C# static class

The C# static class is like the normal class but it cannot be instantiated. It can have only static members. The advantage of static class is that it provides you guarantee that instance of static class cannot be created.

Points to remember for C# static class

- C# static class contains only static members.
- C# static class cannot be instantiated.
- C# static class is sealed.
- C# static class cannot contain instance constructors.

C# static class example

Let's see the example of static class that contains static field and static method.

```
using System;
  public static class MyMath
   {
       public static float PI=3.14f;
       public static int cube(int n){return n*n*n;}
   }
  class TestMyMath{
      public static void Main(string[] args)
       {
           Console.WriteLine("Value of PI is: "+MyMath.PI);
           Console.WriteLine("Cube of 3 is: " + MyMath.cube(3));
       }
   }
```

C# static constructor

C# static constructor is used to initialize static fields. It can also be used to perform any action that is to be performed only once. It is invoked automatically before first instance is created or any static member is referenced.

Points to remember for C# Static Constructor

- C# static constructor cannot have any modifier or parameter.
- C# static constructor is invoked implicitly. It can't be called explicitly.

C# Static Constructor example

Let's see the example of static constructor which initializes the static field rateOfInterest in Account class.

```
using System;
  public class Account
   {
       public int id;
       public String name;
       public static float rateOfInterest;
       public Account(int id, String name)
       {
           this.id = id;
           this.name = name;
       }
```

```
    static Account()
    {
        rateOfInterest = 9.5f;
    }
    public void display()
    {
        Console.WriteLine(id + " " + name+" "+rateOfInterest);
    }
}
class TestEmployee{
   public static void Main(string[] args)
    {
        Account a1 = new Account(101, "Sonoo");
        Account a2 = new Account(102, "Mahesh");
        a1.display();
        a2.display();

    }
 }
```

Output:

```
101 Sonoo 9.5
102 Mahesh 9.5
```

C# Properties

C# Properites doesn't have storage location. C# Properites are extension of fields and accessed like fields.

The Properties have accessors that are used to set, get or compute their values.

Usage of C# Properties

1. C# Properties can be read-only or write-only.
2. We can have logic while setting values in the C# Properties.
3. We make fields of the class private, so that fields can't be accessed from outside the class directly. Now we are forced to use C# properties for setting or getting values.

C# Properties Example

```
using System;
   public class Employee
```

```
  {
      private string name;

      public string Name
      {
         get
         {
            return name;
         }
         set
         {
            name = value;
         }
      }
 }
 class TestEmployee{
     public static void Main(string[] args)
      {
          Employee e1 = new Employee();
          e1.Name = "Sonoo Jaiswal";
          Console.WriteLine("Employee Name: " + e1.Name);

      }
  }
```

Output:

```
Employee Name: Sonoo Jaiswal
```

C# Properties Example 2: having logic while setting value

```
using System;
  public class Employee
   {
      private string name;

      public string Name
      {
         get
         {
            return name;
         }
         set
```

```
        {
            name = value+" JavaTpoint";

        }
    }
}
class TestEmployee{
    public static void Main(string[] args)
    {
        Employee e1 = new Employee();
        e1.Name = "Sonoo";
        Console.WriteLine("Employee Name: " + e1.Name);
    }
 }
```

Output:

```
Employee Name: Sonoo JavaTpoint
```

C# Properties Example 3: read-only property

```
using System;
   public class Employee
    {
        private static int counter;

        public Employee()
        {
            counter++;
        }
        public static int Counter
        {
            get
            {
                return counter;
            }
         }
   }
   class TestEmployee{
       public static void Main(string[] args)
       {
           Employee e1 = new Employee();
           Employee e2 = new Employee();
        Employee e3 = new Employee();
        //e1.Counter = 10;//Compile Time Error: Can't set value

        Console.WriteLine("No. of Employees: " + Employee.Counter);
    }
}
```

Output:

```
No. of Employees: 3
```

C# Base

In C#, base keyword is used to access fields, constructors and methods of base class.

You can use base keyword within instance method, constructor or instance property accessor only. You can't use it inside the static method.

C# base keyword: accessing base class field

We can use the base keyword to access the fields of the base class within derived class. It is useful if base and derived classes have the same fields. If derived class doesn't define same field, there is no need to use base keyword. Base class field can be directly accessed by the derived class.

Let's see the simple example of base keyword in C# which accesses the field of base class.

```
using System;
public class Animal{
    public string color = "white";
}
public class Dog: Animal
{
    string color = "black";
    public void showColor()
    {
        Console.WriteLine(base.color);
        Console.WriteLine(color);
    }

}
public class TestBase
```

```
{
    public static void Main()
    {
        Dog d = new Dog();
        d.showColor();
    }
}
```

Output:

```
white
black
```

C# base keyword example: calling base class method

By the help of base keyword, we can call the base class method also. It is useful if base and derived classes defines same method. In other words, if method is overridden. If derived class doesn't define same method, there is no need to use base keyword. Base class method can be directly called by the derived class method.

Let's see the simple example of base keyword which calls the method of base class.

```
using System;
public class Animal{
    public virtual void eat(){
        Console.WriteLine("eating...");
    }
}
public class Dog: Animal
{
    public override void eat()
    {
        base.eat();
        Console.WriteLine("eating bread...");
    }

}
public class TestBase
{
    public static void Main()
    {
        Dog d = new Dog();
        d.eat();
    }
}
```

Output:

```
eating...
eating bread...
```

C# inheritance: calling base class constructor internally

Whenever you inherit the base class, base class constructor is internally invoked. Let's see the example of calling base constructor.

```
using System;
public class Animal{
    public Animal(){
        Console.WriteLine("animal...");
    }
}
public class Dog: Animal
{
    public Dog()
    {
        Console.WriteLine("dog...");
    }

}
public class TestOverriding
{
    public static void Main()
    {
        Dog d = new Dog();

    }
}
```

Output:

```
animal...
dog...
```

C# Sealed

C# sealed keyword applies restrictions on the class and method. If you create a sealed class, it cannot be derived. If you create a sealed method, it cannot be overridden.

Note: Structs are implicitly sealed therefore they can't be inherited.

C# Sealed class

C# sealed class cannot be derived by any class. Let's see an example of sealed class in C#.

```
using System;
sealed public class Animal{
    public void eat() { Console.WriteLine("eating..."); }
}
public class Dog: Animal
{
    public void bark() { Console.WriteLine("barking..."); }
}
public class TestSealed
{
    public static void Main()
    {
        Dog d = new Dog();
        d.eat();
        d.bark();


    }
}
```

Output:

```
Compile Time Error: 'Dog': cannot derive from sealed type 'Animal'
```

C# Sealed method

The sealed method in C# cannot be overridden further. It must be used with override keyword in method.

Let's see an example of sealed method in C#.

```
using System;
public class Animal{
    public virtual void eat() { Console.WriteLine("eating..."); }
    public virtual void run() { Console.WriteLine("running..."); }

}
public class Dog: Animal
{
    public override void eat() { Console.WriteLine("eating bread..."); }
    public sealed override void run() {
    Console.WriteLine("running very fast...");
    }
}
public class BabyDog : Dog
{
    public override void eat() { Console.WriteLine("eating biscuits..."); }
    public override void run() { Console.WriteLine("running slowly..."); }
}
public class TestSealed
{
    public static void Main()
    {
        BabyDog d = new BabyDog();
        d.eat();
        d.run();
    }
}
```

Output:

```
Compile Time Error: 'BabyDog.run()': cannot override inherited member
'Dog.run()' because it is sealed
```

Note: Local variables can't be sealed.

```
using System;
public class TestSealed
{
    public static void Main()
    {
        sealed int x = 10;
        x++;
        Console.WriteLine(x);
    }
}
```

Output:

```
Compile Time Error: Invalid expression term 'sealed'
```

C# Abstract

Abstract classes are the way to achieve abstraction in C#. Abstraction in C# is the process to hide the internal details and showing functionality only. Abstraction can be achieved by two ways:

1. Abstract class
2. Interface

Abstract class and interface both can have abstract methods which are necessary for abstraction.

Abstract Method

A method which is declared abstract and has no body is called abstract method. It can be declared inside the abstract class only. Its implementation must be provided by derived classes. For example:

```
public abstract void draw();
```

An abstract method in C# is internally a virtual method so it can be overridden by the derived class.

You can't use static and virtual modifiers in abstract method declaration.

C# Abstract class

In C#, abstract class is a class which is declared abstract. It can have abstract and non-abstract methods. It cannot be instantiated. Its implementation must be provided by derived classes. Here, derived class is forced to provide the implementation of all the abstract methods.

Let's see an example of abstract class in C# which has one abstract method draw(). Its implementation is provided by derived classes: Rectangle and Circle. Both classes have different implementation.

```csharp
using System;
public abstract class Shape
{
    public abstract void draw();
}
public class Rectangle : Shape
{
    public override void draw()
    {
        Console.WriteLine("drawing rectangle...");
    }
}
public class Circle : Shape
{
    public override void draw()
    {
        Console.WriteLine("drawing circle...");
    }
}
public class TestAbstract
{
    public static void Main()
    {
        Shape s;
        s = new Rectangle();
        s.draw();
        s = new Circle();
        s.draw();
    }
}
```

Output:

```
drawing ractangle...
drawing circle...
```