

**LAB MANUAL**

**Course: CSC412 – Visual Progra**

**mming**

Logo, company name

Description automatically generated

**Department of Computer Science**

**Learning Procedure**

1. Stage **J** (**Journey inside-out the concept**)
2. Stage **a1** (**Apply the learned**)
3. Stage **v** (**Verify the accuracy**)
4. Stage **a2** (**Assess your work**)

**COMSATS University Islamabad, Wah Campus(CUI)**

i

**Islamabad**

**LAB # 01**

# Statement Purpose:

This lab will give you an introduction to Visual Studio environment and developing first Console based application using C# language.

# Activity Outcomes:

This lab teaches you the following topics:

* + Understanding Visual Studio environment
  + Creating Console based application using Visual Studio
  + Developing first console based application using C# language
  + Understanding structure of console based application

# Instructor Note:

Download and Install Visual Studio 2015 or above. In this lab, Visual Studio 2015 is used.

# Stage J (Journey) Introduction

Visual Studio is a complete set of development tools for building ASP.NET Web applications, XML Web Services, desktop applications, and mobile applications. Visual Basic, Visual C#, and Visual C++ all use the same integrated development environment (IDE), which enables tool sharing and eases the creation of mixed-language solutions. In addition, these languages use the functionality of the .NET Framework, which provides access to key technologies that simplify the development of ASP Web applications and XML Web Services.

Console applications are typically designed without a graphical user interface (GUI) and are compiled into an executable file. You interact with a console application by typing instructions at the command prompt.

# Stage a1 (apply) Lab Activities:

### Activity 1:

Visual Studio environment should be customized for C# Language. By doing this, C# based tools and options would be more visible. Existing Visual Studio environment settings can also be exported.

### Solution:

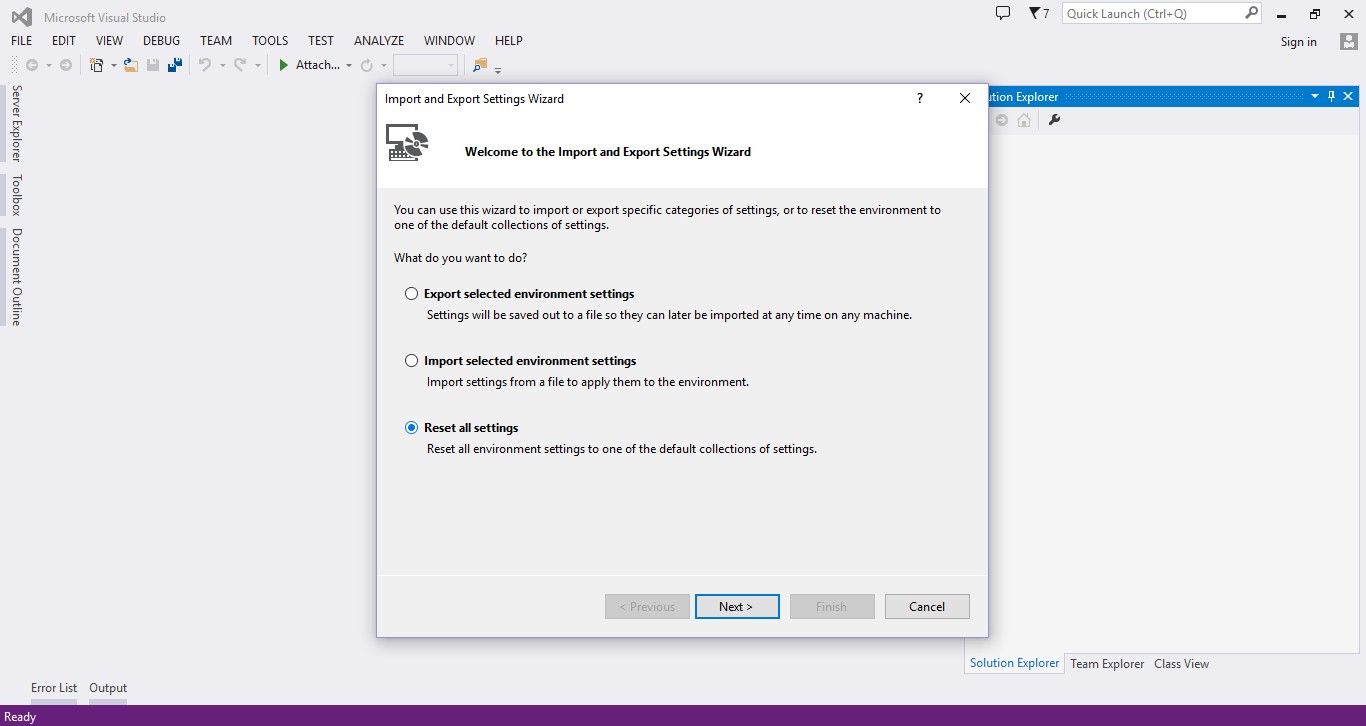
Follow these steps for environment settings:

* + Launch Visual Studio

###### Go in Tools > Import and Export Settings

* + Select **Reset all Settings** and press **Next**

Note: To export current environment settings, choose **Export selected environment settings** option.



### Activity 2:

Developing first console based application in Visual Studio using .net framework.

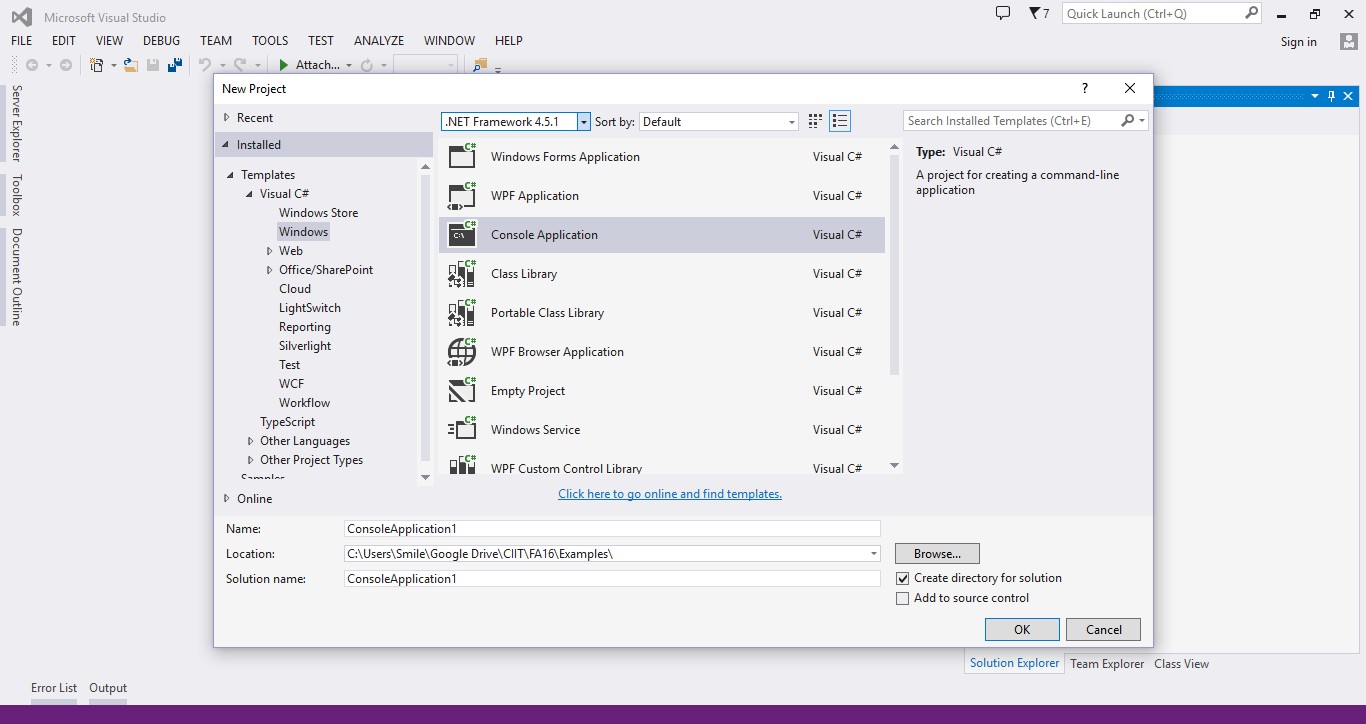
### Solution:

To create a new project, Go to **File > New > Project** or Press **Ctrl+Shift+N**.

A popup will appear showing multiple options. From left tree of the popup, select **Windows** under **Templates > Visual C#**. On right side of the popup, type of Windows based projects are shown which can be developed using Visual C#. A .net framework version can be selected from drop down menu in popup.

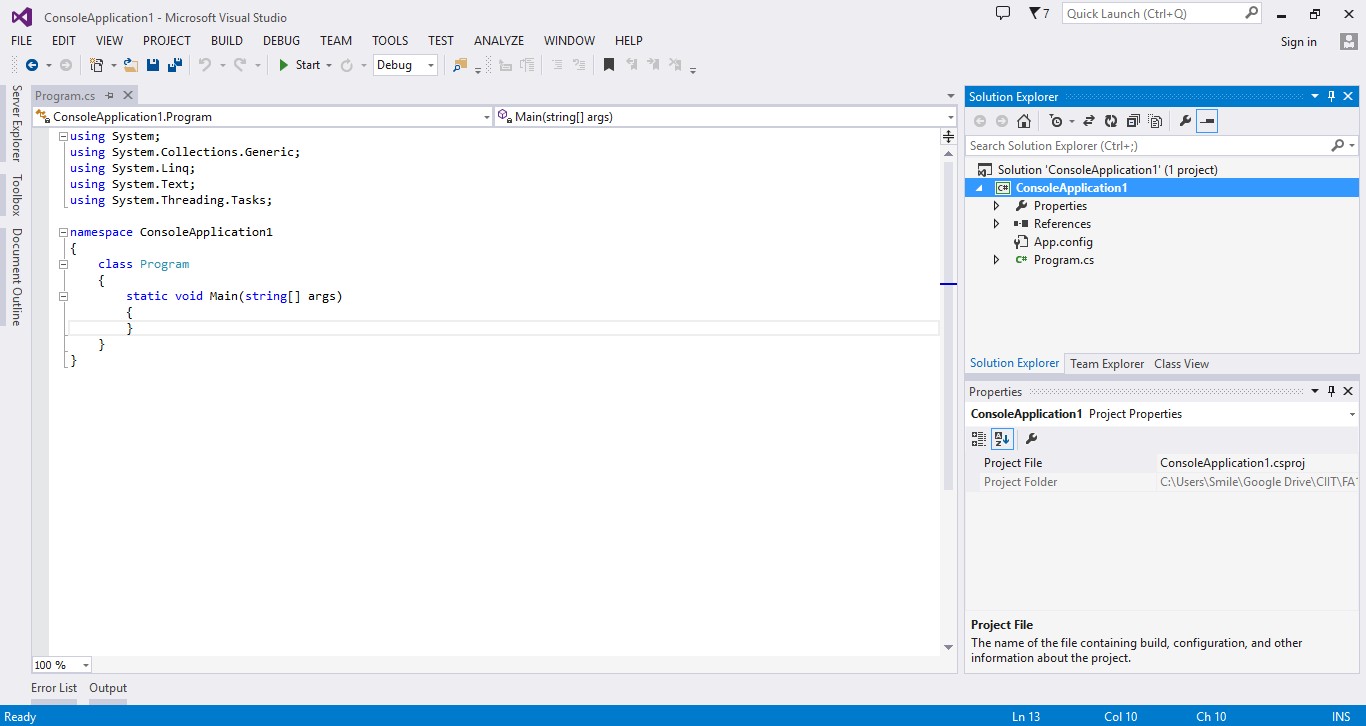
Select Console Application, choose name of project, location where project will be created and project Solution name. Press OK to create the project.

See details of other Template types at [https://msdn.microsoft.com/en-](https://msdn.microsoft.com/en-us/library/0fyc0azh(v%3Dvs.120).aspx) [us/library/0fyc0azh(v=vs.120).aspx](https://msdn.microsoft.com/en-us/library/0fyc0azh(v%3Dvs.120).aspx)



A new console based application will be created from template. Solution Explorer shows the project file and references structure. **App.config** file contains the information about the configuration of the project solution. **Program.cs** file contains the code which will be written by developer.

Compile and run application by clicking **Start** button or **Debug > Start Debugging** or press **F5**.



For now, a blank output screen will appear. In some Visual Studio configurations, output will stop. If it does not stop then write a code line **Console.ReadKey();** inside Main method. By writing this line of code, console will stop to take key input from keyboard.

### Activity 3:

Writing first “Hello World” console based application using Visual C#.

### Solution:

The following procedure creates a C# version of the traditional "Hello World!" program. The program displays the string Hello World!

If Program.cs isn't open in the **Code Editor**, open the shortcut menu for **Program.cs** in

**Solution Explorer**, and then choose **View Code**.

Replace the contents of Program.cs with the following code.

// A Hello World! program in C#. using System;

namespace HelloWorld

{

class Hello

{

static void Main()

{

Console.WriteLine("Hello World!");

// Keep the console window open in debug mode. Console.WriteLine("Press any key to exit."); Console.ReadKey();

}

}

}

Choose the F5 key to run the project. A Command Prompt window appears that contains the line Hello World!

Next, the important parts of this program are examined.

The first line contains a comment. The characters // convert the rest of the line to a comment.

// A Hello World! program in C#.

You can also comment out a block of text by enclosing it between the /\* and \*/ characters. This is shown in the following example.

/\* A "Hello World!" program in C#.

This program displays the string "Hello World!" on the screen. \*/

###### Main Method

A C# console application must contain a Main method, in which control starts and ends. The Main method is where you create objects and execute other methods.

The Main method is a static (C# Reference) method that resides inside a class or a struct. In the previous "Hello World!" example, it resides in a class named Hello. You can declare the Main method in one of the following ways:

It can return **void**.

static void Main()

{

//...

}

It can also return an integer.

static int Main()

{

//... return 0;

}

With either of the return types, it can take arguments.

static void Main(string[] args)

{

//...

}

Or

static int Main(string[] args)

{

//... return 0;

}

The parameter of the Main method, args, is a **string** array that contains the command-line arguments used to invoke the program. Unlike in C++, the array does not include the name of the executable (exe) file.

The call to [ReadKey](https://msdn.microsoft.com/en-us/library/system.console.readkey(v%3Dvs.120).aspx) at the end of the Main method prevents the console window from closing before you have a chance to read the output when you run your program in debug mode, by pressing F5.

###### Input and Output

C# programs generally use the input/output services provided by the run-time library of the .NET Framework. The statement System.Console.WriteLine("Hello World!"); uses the WriteLine method. This is one of the output methods of the Console class in the run-time library. It displays its string parameter on the standard output stream followed by a new line. Other Console methods are available for different input and output operations. If you include the using System; directive at the beginning of the program, you can directly use the System classes and methods without fully qualifying them. For example, you can call Console.WriteLine instead of System.Console.WriteLine:

using System; Console.WriteLine("Hello World!");

### Activity 4:

The Main method is the entry point of a C# console application or windows application. (Libraries and services do not require a Main method as an entry point.). When the application is started, the Main method is the first method that is invoked.

Note: There can only be one entry point in a C# program. If you have more than one class that has a Main method, you must compile your program with the **/main** compiler option to specify which Main method to use as the entry point

This activity is about passing and accessing command line arguments.

### Solution:

Following example prints the number of command line arguments on console.

class TestClass

{

static void Main(string[] args)

{

// Display the number of command line arguments: System.Console.WriteLine(args.Length);

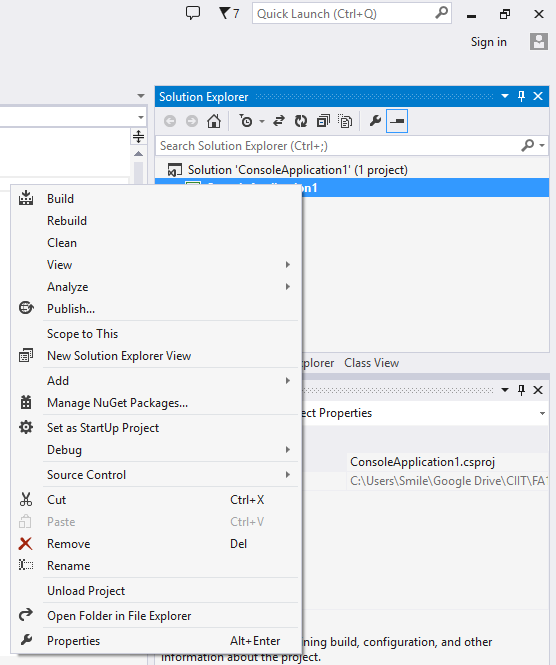
}

}

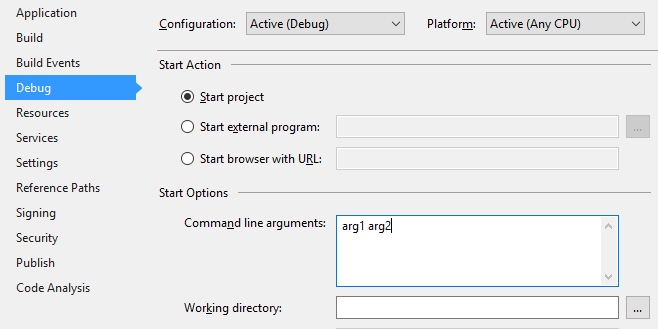
To pass command line arguments to the program, it can be done while executing the program from command line. Each argument should be separated by a space.

For example, If output file is executed from command line C:\TestClass\bin\Debug\TestClass.exe arg1 arg2 Then the output of above program would be 2

Command line arguments can also be specified in project properties. To open project properties, go to Project > *ProjectName* Properties or right click on Project name in Solution Explorer and click on Properties or press Alt+Enter



In project properties, go to Debug section and specify Command Line Arguments separated with space.



Press F5 to run the program and output will be 2.

Array indexes can be used to access the contents of command line arguments. Following example will print the contents of command line arguments on console:

class Program

{

static void Main(string[] args)

{

if (args.Length > 0)

{

Console.WriteLine(args[0]); Console.WriteLine(args[1]);

}

Console.ReadKey();

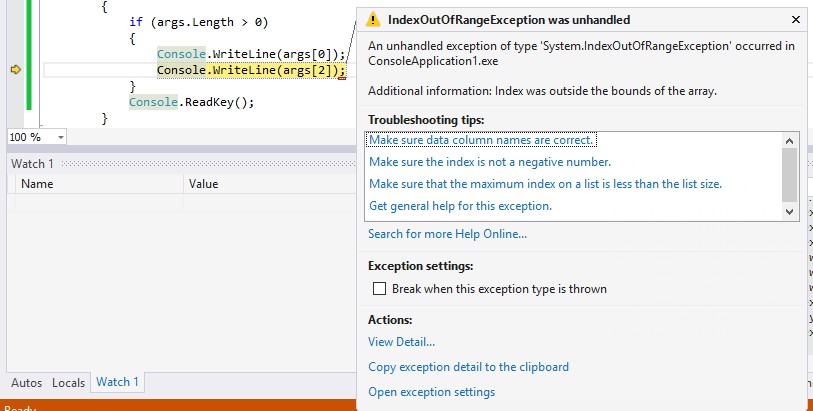
}

}

Output of the program is:

arg1 arg2

Note: IndexOutOfRange exception would be thrown if an index does not contain command line argument. For example, in above example, if arg1 or arg2 are not passed as command line argument then the above code would through IndexOutOfRanage Exception.



###### Using foreach loop

Another approach to iterating over the array is to use the foreach statement as shown in this example. The **foreach** statement can be used to iterate over an array, a .NET Framework collection class, or any class or struct that implements the IEnumerable interface.

class CommandLine2

{

static void Main(string[] args)

{

Console.WriteLine("Number of command line parameters = {0}", args.Length);

foreach (string s in args)

{

Console.WriteLine(s);

}

}

}

/\* Output:

Number of command line parameters = 3 John

Paul Mary

\*/

###### Using for loop

This example displays the command line arguments passed to a command-line application. The output shown is for the first entry in the table above.

class CommandLine

{

static void Main(string[] args)

{

// The Length property provides the number of array elements Console.WriteLine("parameter count = {0}", args.Length);

for (int i = 0; i < args.Length; i++)

{

Console.WriteLine("Arg[{0}] = [{1}]", i, args[i]);

}

}

}

/\* Output (assumes 3 cmd line args): parameter count = 3

Arg[0] = [a]

Arg[1] = [b]

Arg[2] = [c]

\*/

# Stage v (verify) Home Activities:

### Activity 1:

Develop a console based application using Visual C# to print your name, registration number and address on console.

### Activity 2:

Develop a console based application which takes command-line arguments and print them on console.

Test your application on following command line arguments:

|  |  |
| --- | --- |
| **Input on Command-line** | **Array of strings passed to Main** |
| **consoleApp.exe a b c** | "a"  "b"  "c" |
| **consoleApp.exe one two** | "one"  "two" |
| **consoleApp.exe "one two" three** | "one two" "three" |

### Activity 3:

Console.ReadLine method reads the next line of characters from the standard input stream. Read following article at MSDN and practice using ReadLine method: https://msdn.microsoft.com/en-us/library/system.console.readline