Information Processing Techniques

**Lab 1**

# Introduction to C#

# and

# Windows Form Application



29/08/2023

Programming in C#

**Section # 1: Console Applications**

**Introduction:**

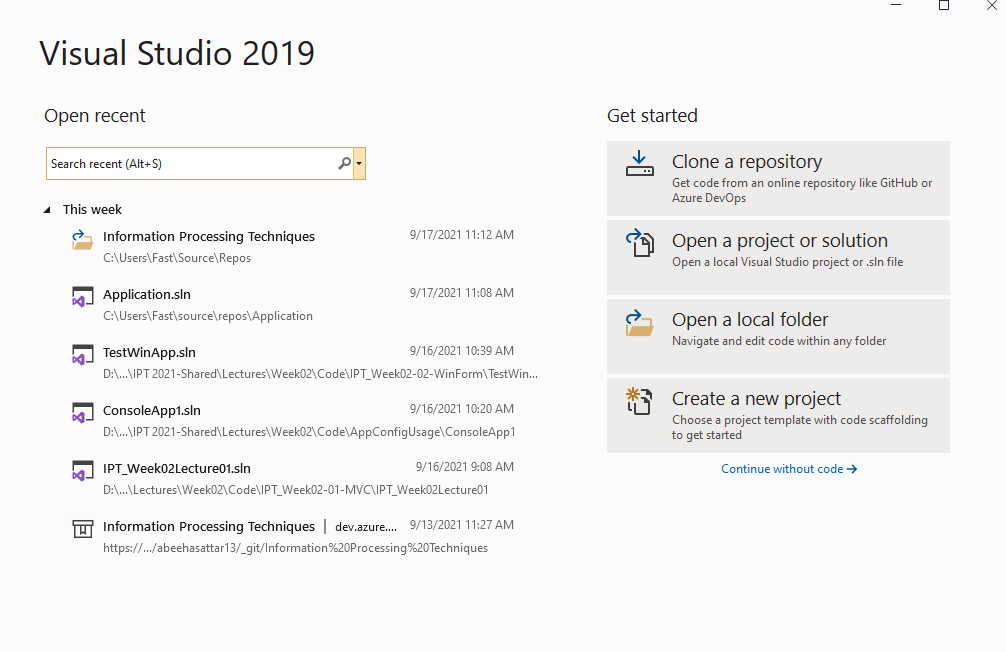
The C# programming language (pronounced "C sharp") was introduced with the .NET Framework. It represents a milestone in the area of programming language development. Even though the genesis of the new C# language is in existing programming languages such as Java and C++, it represents the most innovative, modern, and even—in most areas—the preferred .NET programming language. C# was introduced to combine the key values of Visual Basic and C++. Whereas Visual Basic had ease of use and high developer productivity, C++ represented the ultimate flexibility and control.

**Integrated Development Environment (IDE)**

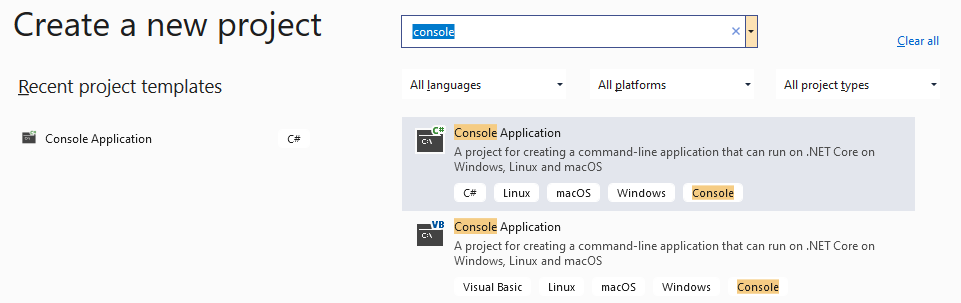
An Integrated Development Environment (IDE) is a tool that helps you write your programs. Microsoft provides Visual Studio to write C# code. Now let’s set up the development environment:

Follow the following steps to get started with C#:

**Step#1: Open Visual Studio and Create a New Project:**

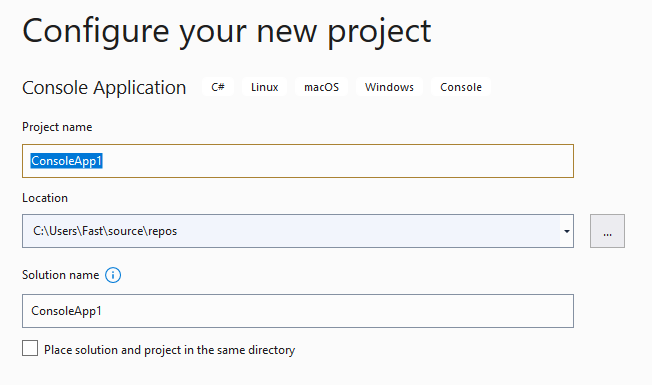


**Step#2: Select the Type of the Project that you wish to create:**



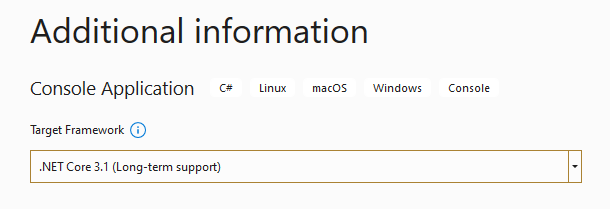
**Step#3: Name your Application:**

When you create a project, a solution file will be created against it. It will share the project name by default, unless you specifically change it.

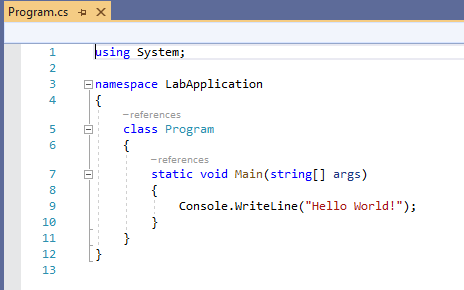


**Step#4: Choose the Framework:**

You can select the framework and then click create to create the project.

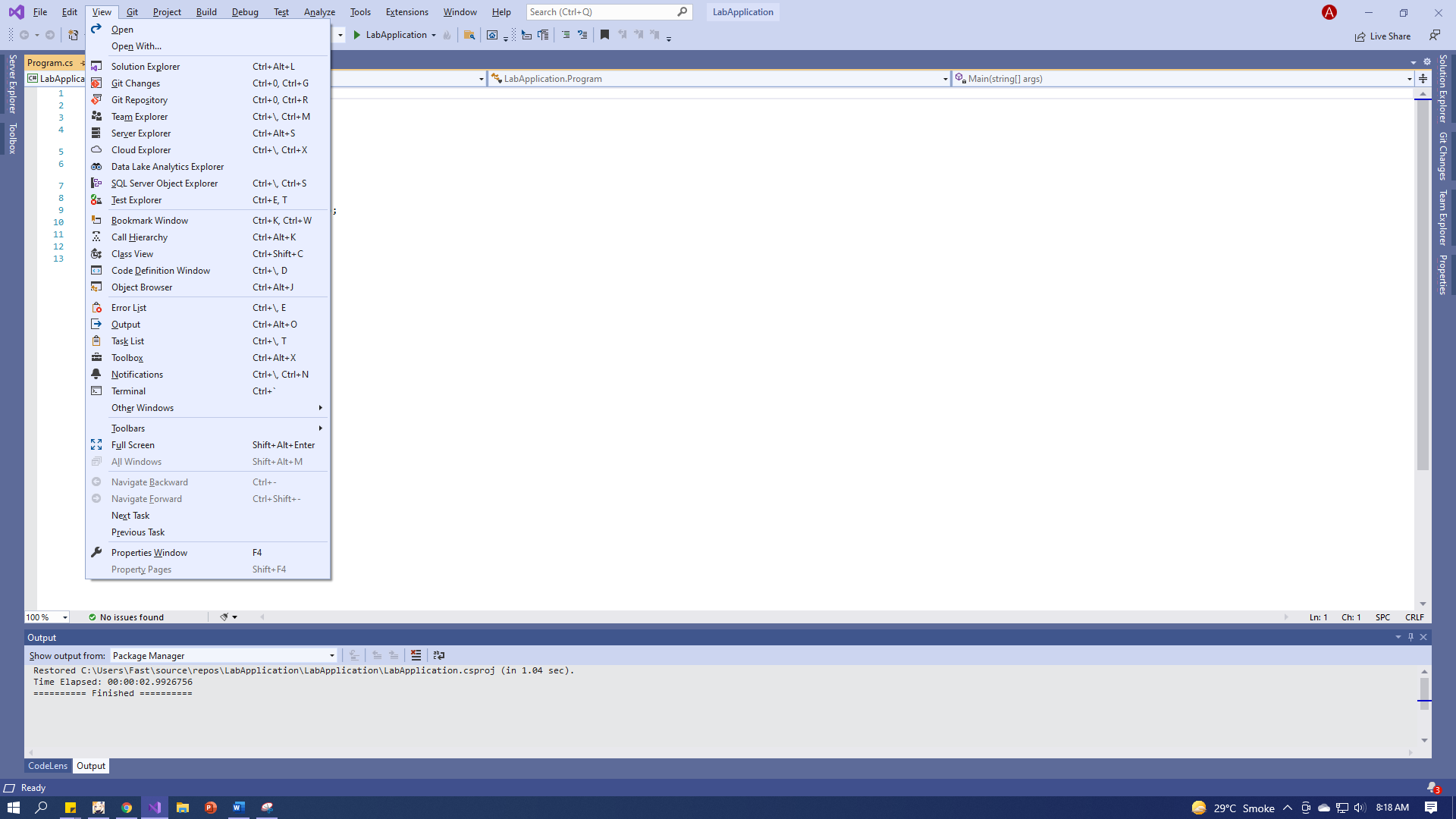


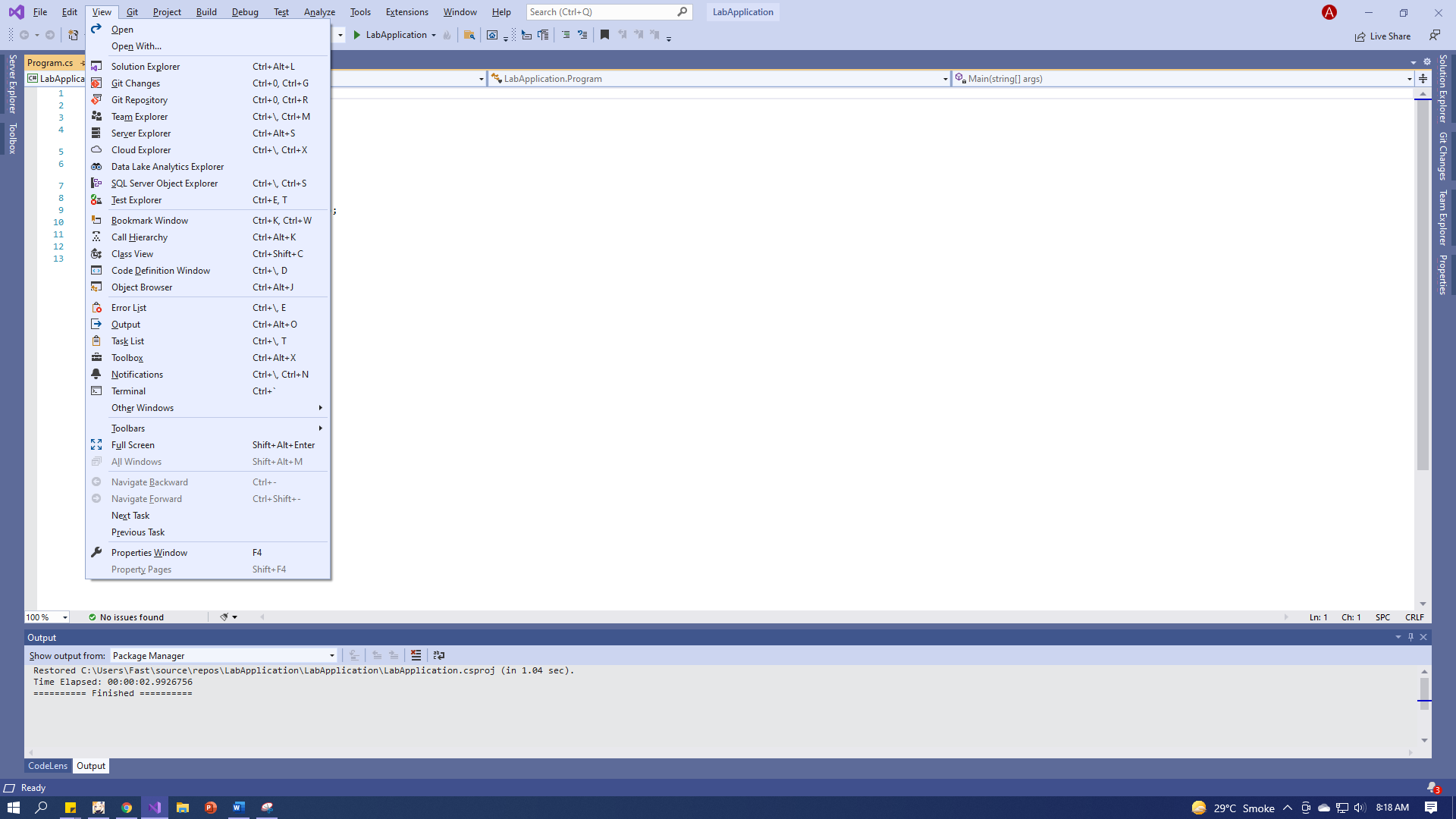
After you create the project, a **program.cs** file will be automatically created for the main project. It contains a very basic **hello world** program.



**Solution Explorer:**

The solution explorer is usually located on the right side of the screen, but you can also access it through the **view** menu.



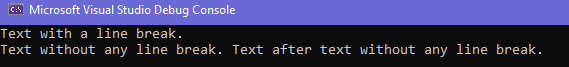


**Standard Input/Output Operations:**

**Output:**

For standard output we have Console.WriteLine() and Console.Write() the difference between two is Console.WriteLine() will automatically move to next line after printing whereas Console.Write() will stay in the same line.

Console.WriteLine(“Text with a line break.”);   
Console.Write(“Text without any line break. “);  
Console.WriteLine(“Text after text without any line break.”);



**Input:**

For standard output we have the following three functions:

1. **Console.Read();** Reads the next character from the standard input.
2. **Console.ReadLine();** Reads the next lines of character from the standard input.
3. **Console.ReadKey();** Read only one character usually used then you give used an option like press Y or N to continue etc.

The basic difference is that Console.Read give int value but that value will be the ASCII value of that. On the other side, Console.ReadLine gives the string as it is given in the input stream.

**NOTE:** both Console.Read and Console.ReadLine is used to get character input to take input in other data types we first need to convert using CONVERT class.

**Example Code:**

int num, intNum;

string Name;

Decimal floatNum;

Console.Write(“Enter string : “);

Name = Console.ReadLine();

Console.Write(“Enter integer : “);

intNum = Convert.ToInt32(Console.ReadLine());

Console.Write(“Enter float : “);

floatNum = Convert.ToDecimal(Console.ReadLine());

Console.Write(“Enter character to get ASCII : “);

num = Console.Read();

Console.WriteLine();

Console.WriteLine(“OUTPUT”);

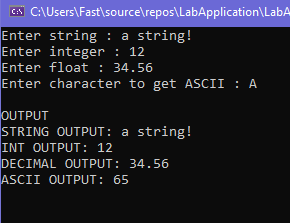
Console.WriteLine(String.Format(“STRING OUTPUT: {0}”, Name));

Console.WriteLine(String.Format(“INT OUTPUT: {0}”, intNum));

Console.WriteLine(String.Format(“DECIMAL OUTPUT: {0}”, floatNum));

Console.WriteLine(String.Format(“ASCII OUTPUT: {0}”, num));

Console.ReadKey();



**Supported Data Types:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reserved Word** | **.NET Type** | **Type** | **Size (bits)** | **Range (values)** |
| byte | Byte | Unsigned integer | 8 | 0 to 255 |
| sbyte | SByte | Signed integer | 8 | -128 to 127 |
| short | Int16 | Signed integer | 16 | -32,768 to 32,767 |
| ushort | Uint16 | Unsigned integer | 16 | 0 to 65,535 |
| int | Int32 | Signed integer | 32 | -2,147,483,648  to  2,147,483,647 |
| uint | Uint32 | Unsigned integer | 32 | 0 to 4294967295 |
| long | Int64 | Signed integer | 64 | -9,223,372,036,854,775,808 to  9,223,372,036,854,775,807 |
| ulong | Uint64 | Unsigned integer | 64 | 0 to 18,446,744,073,709,551,615 |
| float | Single | Single-precision floating point type | 32 | -3.402823e38  to  3.402823e38 |
| double | Double | Double-precision floating point type | 64 | -1.79769313486232e308  to  1.79769313486232e308 |
| decimal | Decimal | Precise fractional or integral type that can represent decimal  numbers with 29 significant digits | 128 | (+ or -)1.0 x 10e-28 to 7.9 x 10e28 |
| char | Char | A single Unicode character | 16 | Unicode symbols used in text |
| bool | Boolean | Logical Boolean type | 8 | True or False |
| object | Object | Base type of all other types |  |  |
| string | String | A sequence of characters |  |  |
| DateTime | DateTime | Represents date and time |  | 0:00:00am 1/1/01 to 11:59:59pm  12/31/9999 |

**Enumerations**

C# provides the enumerations programming construct, which provides a human-readable form of a series of related constant values.

enum Days

{

Monday = 0, Tuesday = 1, Wednesday = 2, Thursday = 3,

Friday = 4, Saturday = 5, Sunday = 7

};

static void Main(string[] args)

{

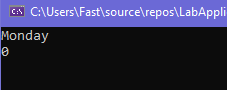
Days day = Days.Monday;

Console.WriteLine(day);

Console.WriteLine((int)day);

Console.ReadKey();

}



**Conditional Statements**

C# offers the conditional statements that we are used to in programming languages like C, C++ and Java. The conditional statements are:

* If
* If…else
* Switch statements

|  |  |
| --- | --- |
| int num;  Console.Write("Enter Number : "); num = Convert.ToInt32(Console.ReadLine()); if(num%2 == 0)  {  Console.WriteLine("Number is Even");  } else  {  Console.WriteLine("Number is Odd");  } | int num;  Console.Write("Enter Number : "); num = Convert.ToInt32(Console.ReadLine());  switch(num % 2)  {  case 0:  {  Console.WriteLine("Number is Even");  break;  }  case 1:  {  Console.WriteLine("Number is Odd"); break;  }    } |

**Loops**

C# provides following types of loop to handle looping requirements.

* For loop
* While loop
* Do while loop
* For each loop

**For each Loop** is used to iterate elements in a collection.

Char[] myArray = {'H', 'e', 'l', 'l', 'o', 'w', 'o', 'r', 'l', 'd'};

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

Console.WriteLine(myArray[i]);

}

Console.ReadKey();

**Methods**

A method is a group of statements that together perform a task, the syntax for defining a method in C# is as follows:

<Access Specifier> <Return Type> <Method Name> (Parameter List) {

Method Body

}

**Access Modifiers**

Access modifiers (access specifiers) describes the scope of an Object and its members. C# provides following access modifiers.

* Public
* Private
* Protected
* Internal
* Protected Internal
* Private Protected

**Public**

It is the most common access specifier in C# which allow access from anywhere.

**Private**

The scope of the accessibility is limited only inside the classes or struct in which they are declared.

**Protected**

Its scope is limited within the class or struct and the class derived (Inherited) from this class.

**Internal**

This can access within the program that contain its declarations and also access within the same assembly level but not from another assembly.

**Protected Internal**

It is same as both protected and internal. It can access anywhere in the same assembly and in the same class also the classes inherited from the same class.

**Private Protected**

It is both private and protected. The scope of the accessibility is limited only inside the classes or struct, and the class derived from this class.

**Classes**

A class definition starts with the access specifier, followed by keyword class then the class name; and the class body enclosed by a pair of curly braces. Following is the code example of the class.

|  |
| --- |
| class Rectangle {  public double length; // Length of a Rectangle  public double breadth; // Breadth of a Rectangle    public Rectangle(double l,double b)  {  length = l;  breadth = b;  }  public double area()  {  return length \* breadth ;  } }  static void Main(string[] args)  {  Rectangle rec = new Rectangle(5, 8);  Console.WriteLine(String.Format("Area Of Rectangle{0}",rec.area()));  Console.ReadKey();  } |

**Section # 2: Windows Form Application**

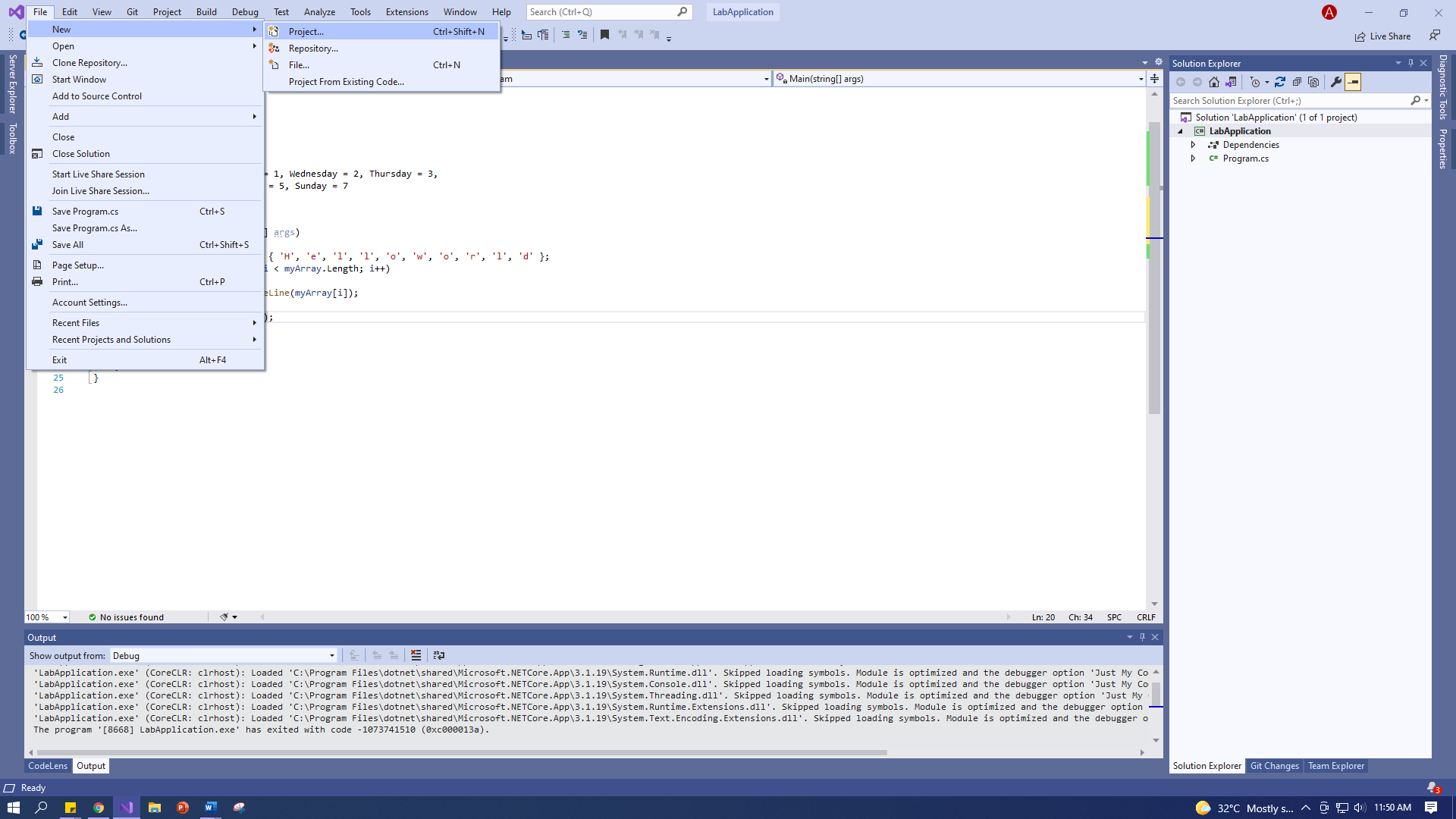
**Introduction:**

A Windows forms application is one that runs on the desktop or laptop computer. A Windows forms application will normally have a collection of controls such as labels, textboxes, buttons, menus, list boxes, etc.

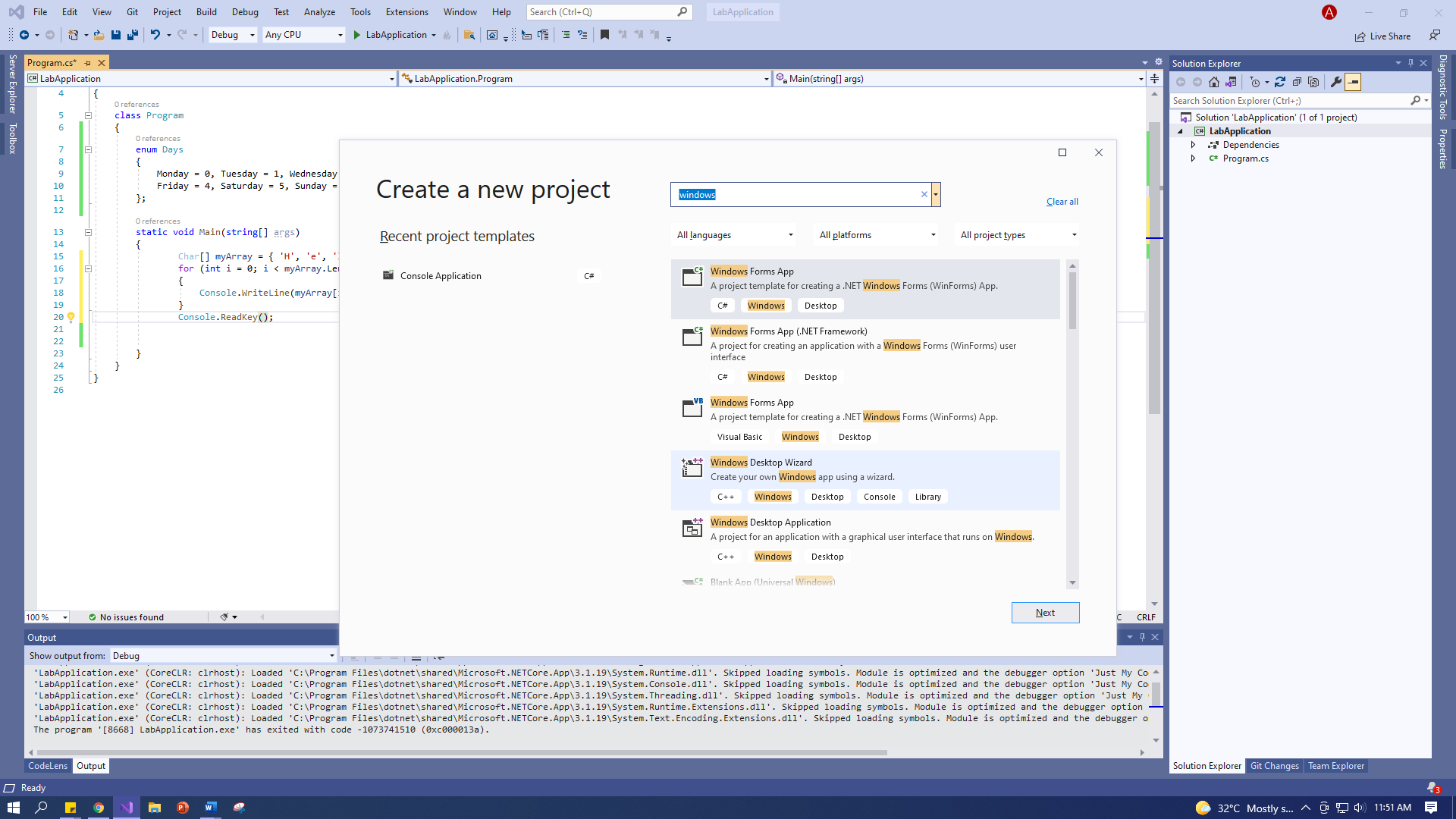
**Creating a Windows Form Application:**

**Step#1: Open Visual Studio and Create a New Project:**

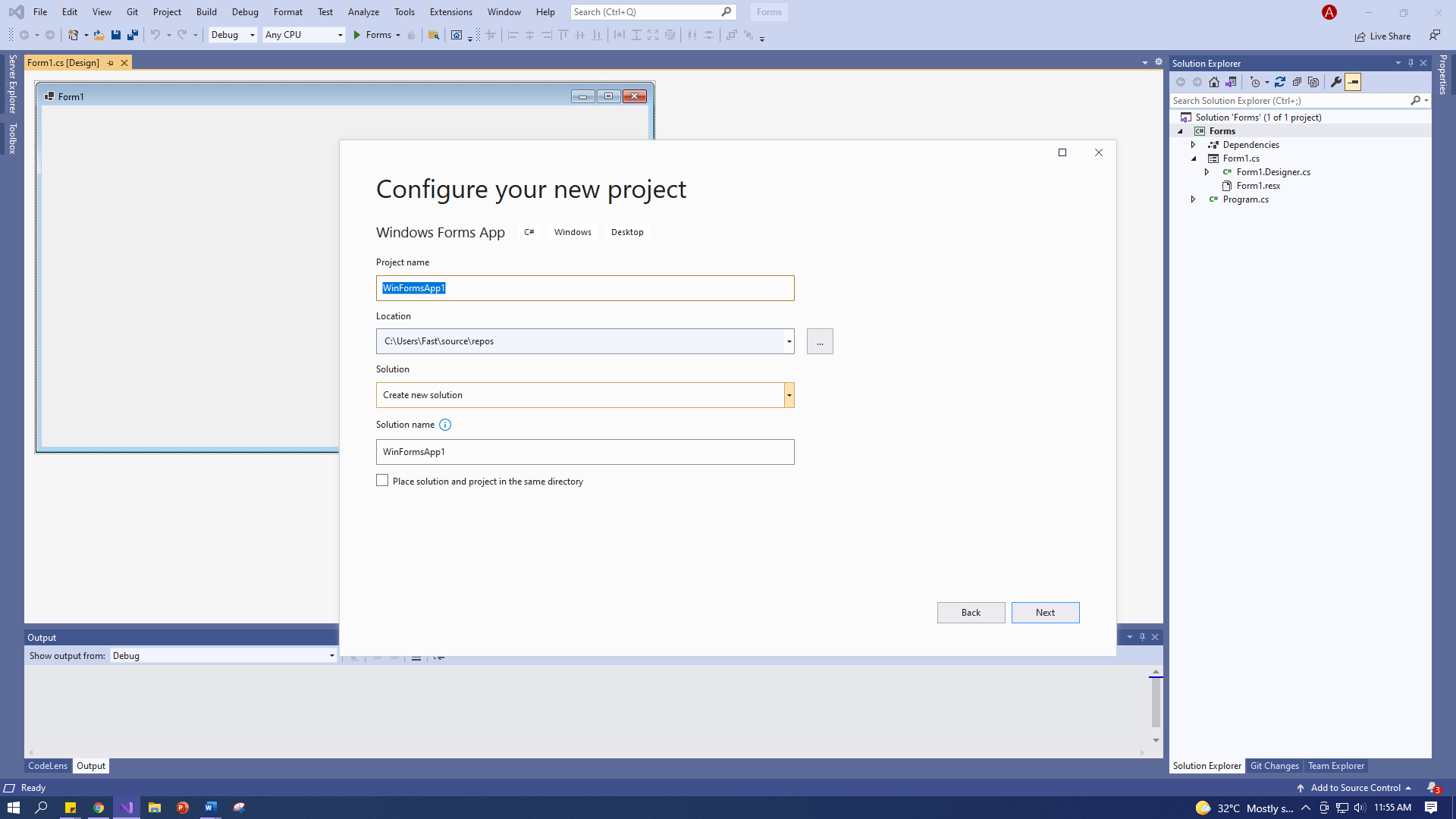
Assuming you already have Visual Studio open, you may go to File > New > Project to create a new project.



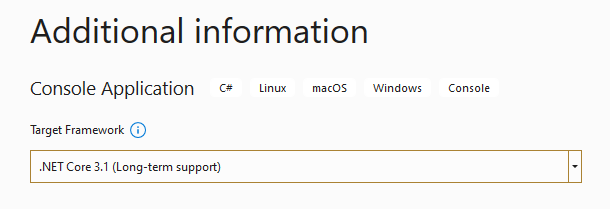
**Step#2: Select the Type of Project that you wish to Create:**



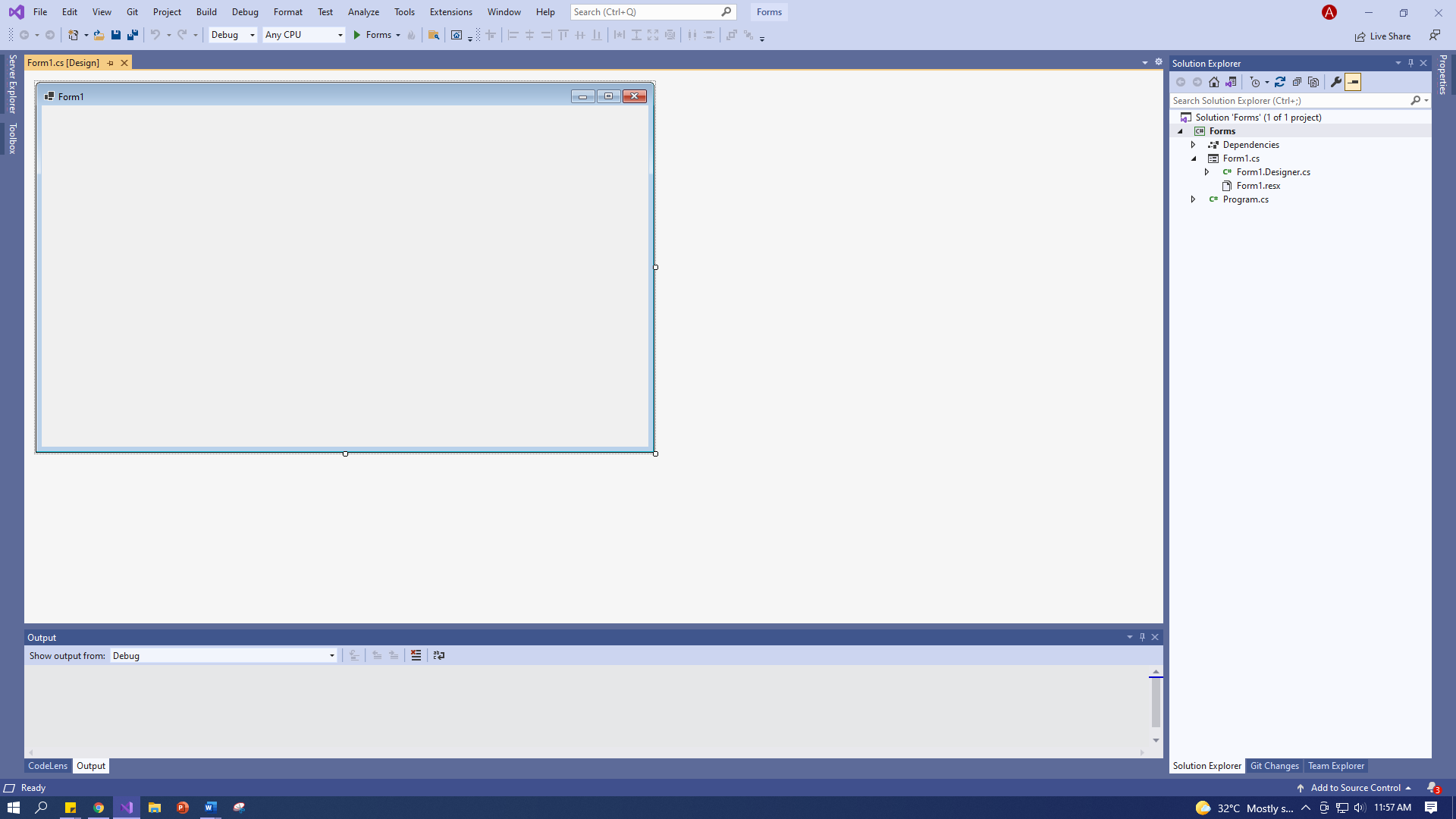
**Step#3: Name Your Project:**



**Step#4: Choose the Framework:**

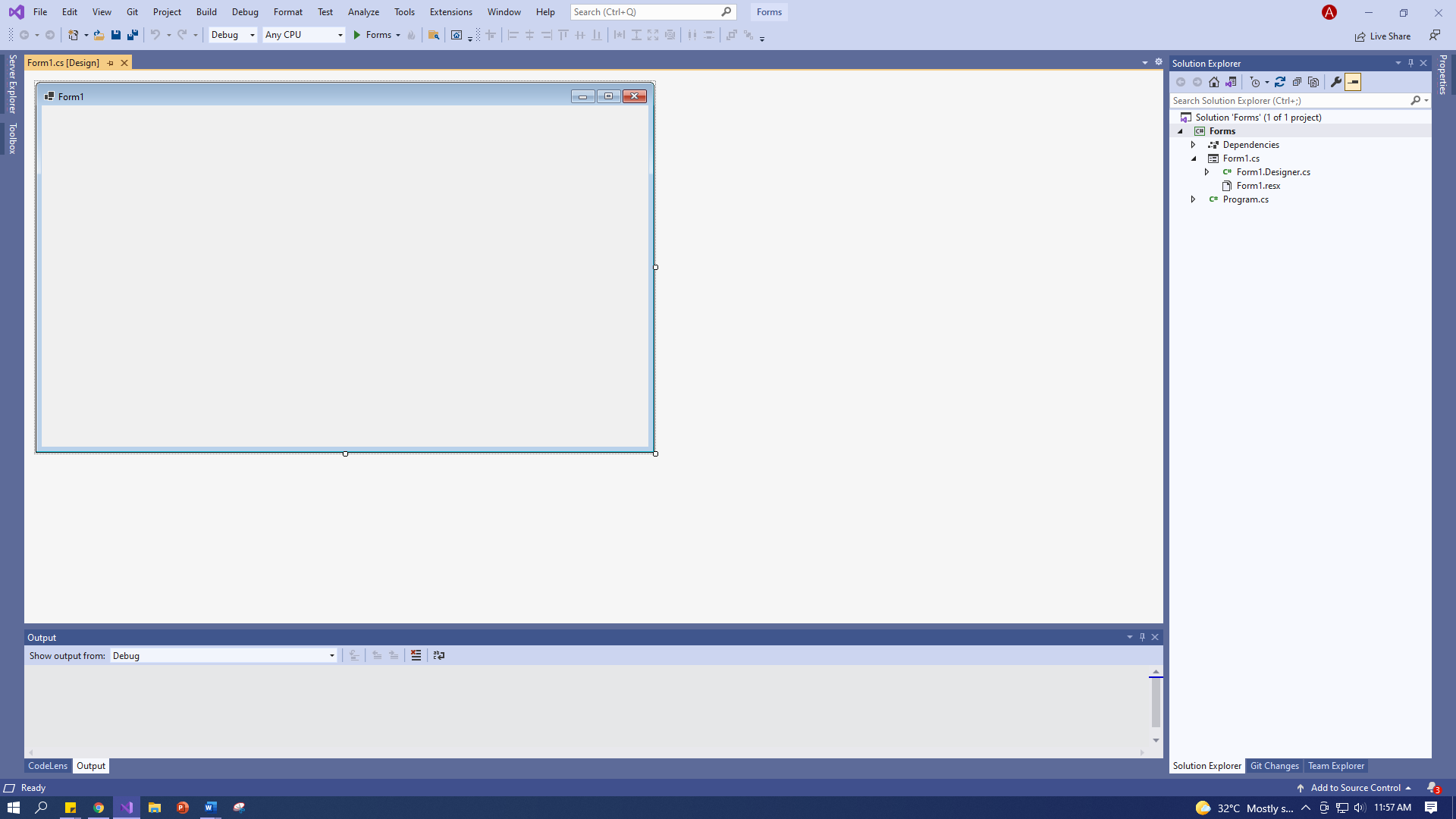
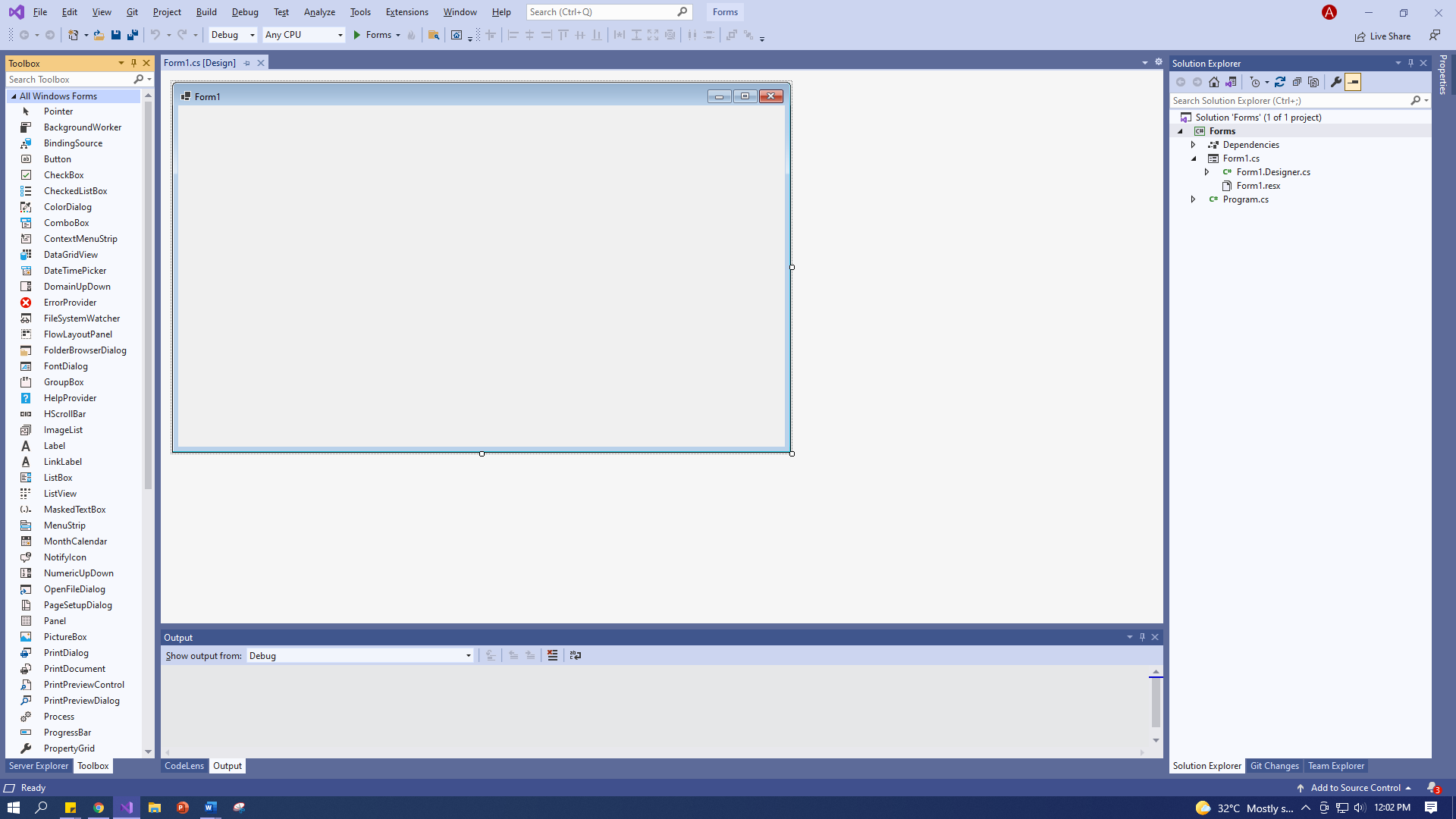


After the project is created successfully you will see an empty Windows form.

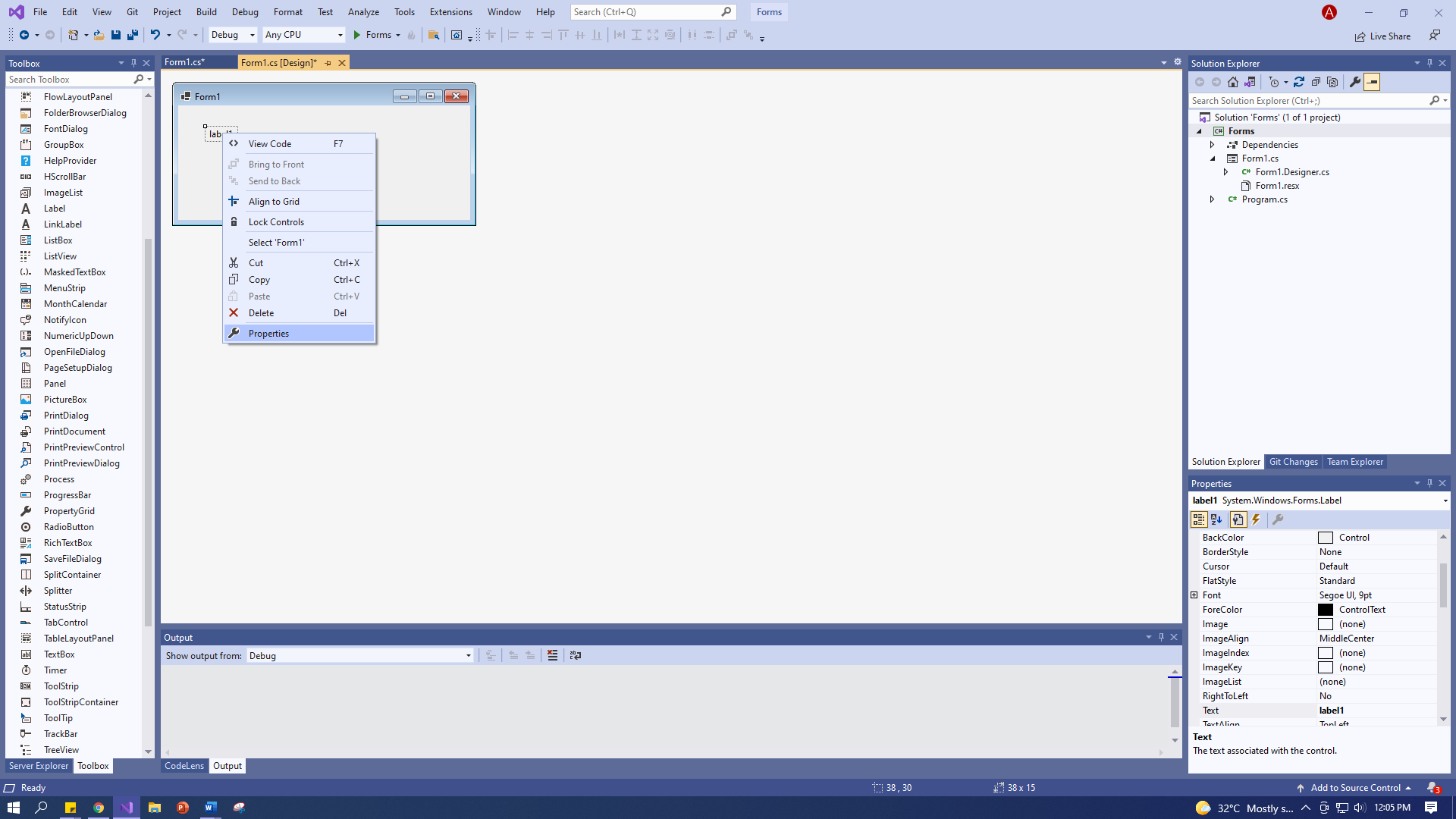
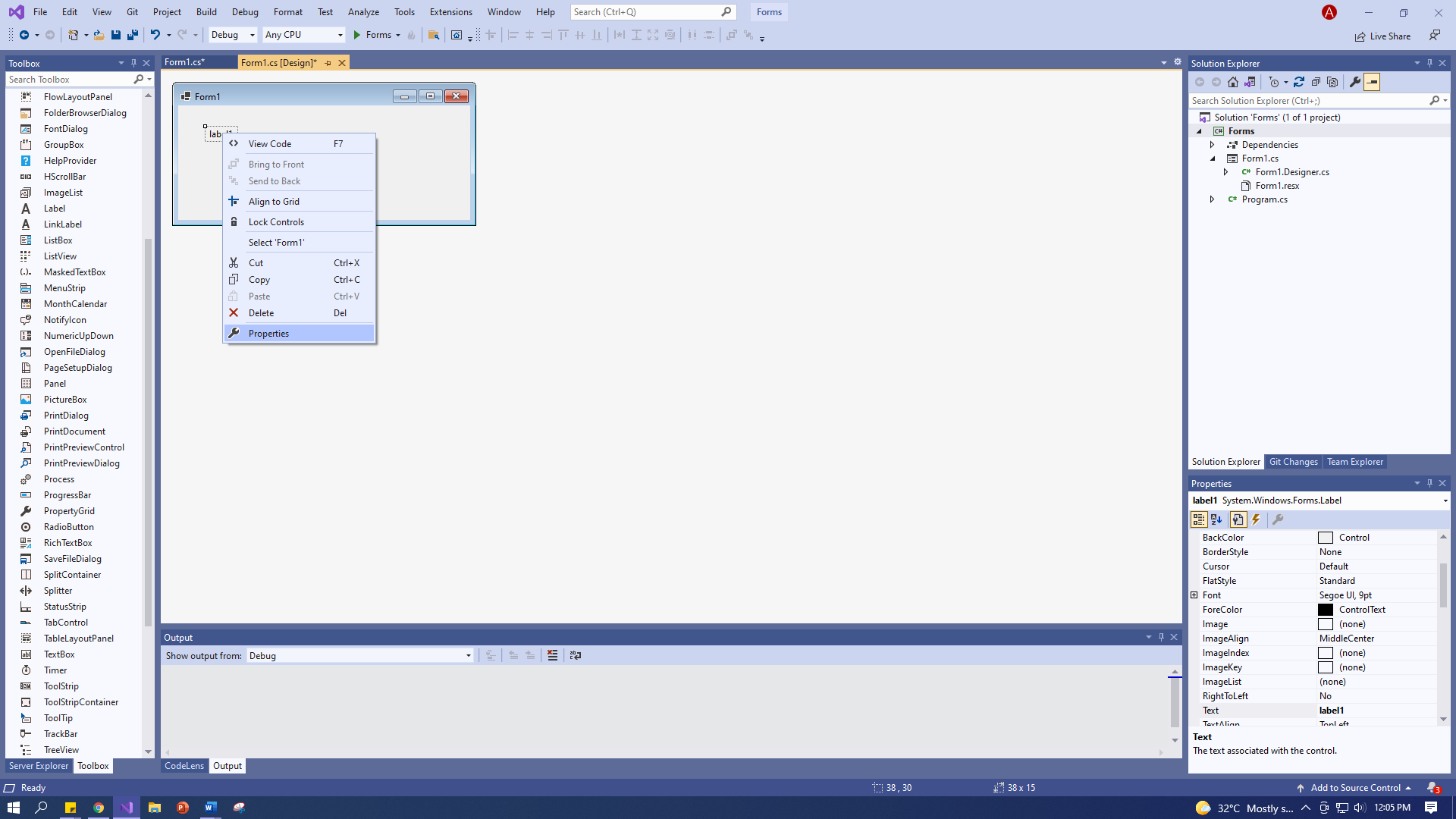


**Step#5: Add Buttons and Fields According to Your Requirements:**

You can open the toolbox from an option on the left side of the screen. Once you open it, you may pin it using the pin icon, and drag and drop buttons and fields from it onto the form.

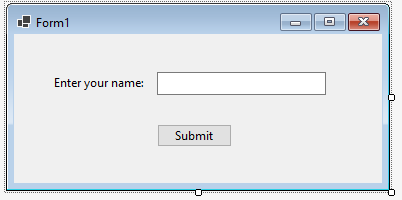
 

You can right click on the labels/buttons you add, to select properties for that field.

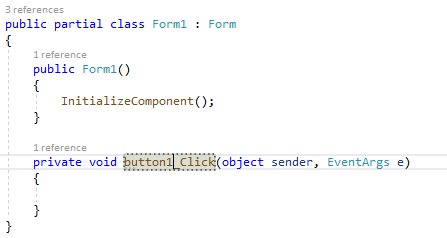
You can change the text of the fields using the “Text” property on the properties tab.

For now, let us add a text box and buttons to the Windows form:



**Step#6: Add Functionality to Your Buttons and Fields:**

Upon double clicking on the buttons and textbox, Visual Studio will take you to the code file for the application. Double clicking on a button also auto-generates the click event of the button.



You can now proceed to write the functionality that you want to implement when the user presses the Submit button.

private void button1\_Click(object sender, EventArgs e)

{

String message = "Welcome ";

message = message + textBox1.Text;

//this will show the message inside the textbox

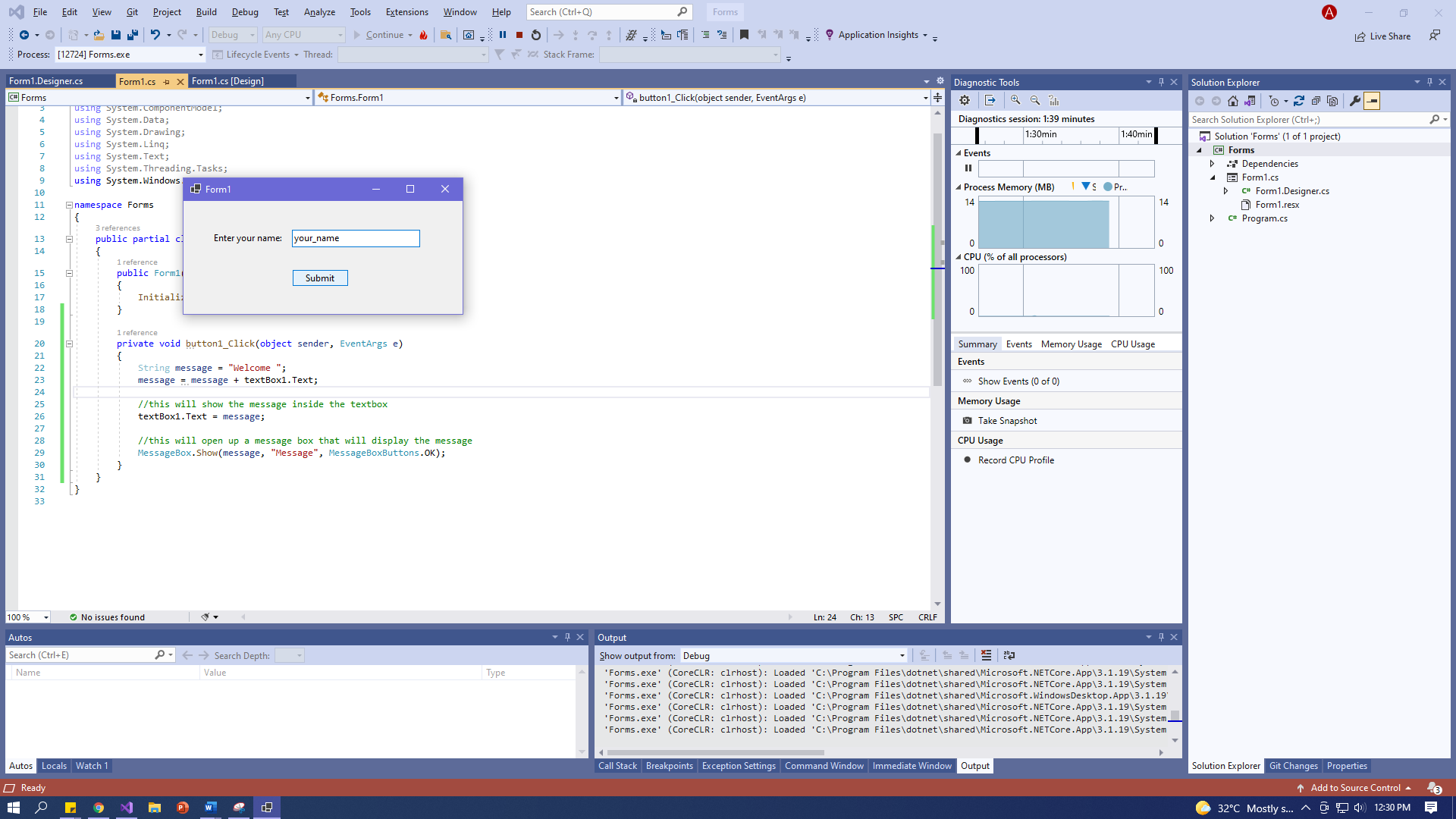
textBox1.Text = message;

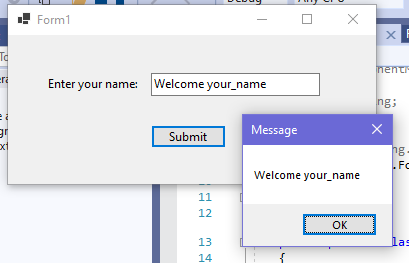
//this will open up a message box that will display the message

MessageBox.Show(message, "Message", MessageBoxButtons.OK);

}

**Step#7: Compile and Run Your Windows Form Application**





**Practice Questions**

**Instructions:**

1. All questions should be Windows Console Applications or Windows Form Applications using C# (on Visual Studio)
2. Solutions of each questions should be uploaded on Google classroom by **11:59 PM**
3. Create separate solution files for each question. Each question should be named as:   
   **k<year><studentID>\_Q<questionNumber>**  
   For example: **k122034\_Q1**
4. Submitted code should have all of the files and can be compiled.

**Task#1:**

Write a program that takes 3 command line arguments, then adds them with an add function, then displays the result in the following format:

**Output:**

1 + 2 + 3 = 6

**Task#2:**

Write a C# program which will accept string input from user and create a new string that is in reverse order, and has all of its letter capitalized.

**Task#3:**

Using 2-D arrays, write a program that allows the user to input two, 3x3 matrices. The program, display the matrices, add them, and output the resultant matrix.

**Input:**

Can be any random numbers

**Ouput:**

The output should follow the highlighted format:

1 2 3

4 5 6

7 8 9

**Task#4:**

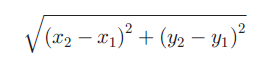
A phone company wants your help to create a telephone directory. Using classes, store the following information that they want to manage:

* Person Name
* Area Code
* Telephone Number

Create a class called TelephoneRecord to store the data. Ask the user about how many records they want to create, then create an array to store the information. The program should allow the user to, add, display and update the information through a menu. You should also implement proper functions for adding, displaying and updating records.

**Task#5:**

The following formula gives the distance between two points, (x1, y1) and (x2, y2) in the Cartesian plane:



Given the center and a point on the circle, you can use this formula to find the radius of the circle. Create a Windows Form Application that asks the user to input the center (x1 and y1) and a point (x2 and y2) on the circle. Your program should then display the following:

1. **Distance**: Use the formula above to create the distance.
2. **Radius**: Same as above. Distance = Radius.
3. **Diagonal**: If r is the radius, then diagonal is 2r.
4. **Circumference**: If r is the radius, the circumference is 2πr.
5. **Area**: If r is the radius, the area is πr2.

You can display the results in the same window, or create a message box to display the results.