



**GLA**  
**UNIVERSITY**  
**MATHURA**  
Recognised by UGC Under Section 2(f)

Accredited with **A<sup>+</sup>** Grade by **NAAC**  
3.46 Score

**12-B Status from UGC**

---

**SESSION 2023-2024**

**MCA –III<sup>rd</sup> Semester**

**Practical File – .NET FRAMEWORK Using C#**

Name – Vishnu Chuadhary

Section – A2

University Roll No. – 2284200238

# INDEX

- 1. Installation of visual studio step by step**
- 2. Creating a simple .net platform program with classes library and functions.**
- 3. Conditional statement in c#**
- 4. Loops statements in C#**
- 5. Classes in C#**
- 6. Delegates in C#**
- 7. File Handling in C#**
- 8. Create a windows form app individual studio with C#**
- 9. Assembly in .net**
- 10. Window based calculator in C#**

### **Que1. Installation of visual studio step by step**

#### **1. Download Visual Studio:**

- Visit <https://visualstudio.microsoft.com/>.
- Select your preferred edition (e.g., Community).
- Download the installer.

#### **2. Run Installer:**

- Execute the downloaded installer.

#### **3. Customize Installation:**

- Choose development workloads (e.g., ".NET desktop," "ASP.NET," etc.).
- Optionally select individual components based on your preferences.

#### **4. Start Installation:**

- Click "Install" to begin the installation process.
- Wait for the installation to complete.

#### **5. Launch and Setup:**

- Launch Visual Studio.
- Sign in or create a Microsoft account (optional).
- Choose default development settings.
- Start using Visual Studio for your projects.

These five steps cover the essential stages of downloading, installing, customizing, and setting up Visual Studio for your development needs.

### **Que2. Creating a simple .net platform program with classes library and functions.**

#### **Step 1: Create a Class Library**

### **1. Open Visual Studio:**

- Launch Visual Studio.

### **2. Create a Class Library Project:**

- Select "Create a new project."
- Choose "Class Library" under the "Class Library" category in the language of your choice (e.g., C#).
- Name the project (e.g., MathLibrary) and click "Create."

### **3. Define a Class and Function:**

- Open the default class file (e.g., Class1.cs).
- Replace the content with a simple math function:

### **4. Build the Class Library:**

- Build the solution to ensure there are no errors.

## **Step 2: Create a Console Application**

### **1. Add a Console Application to the Solution:**

- Right-click on the solution in Solution Explorer.
- Choose "Add" -> "New Project."
- Select "Console App" under the language of your choice (e.g., C#).
- Name the project (e.g., ConsoleApp) and click "Create."

### **2. Reference the Class Library:**

- Right-click on the console application project.
- Choose "Add" -> "Reference."
- Select the class library project (MathLibrary) and click "OK."

### **3. Use the Class Library in the Console Application:**

- Open the Program.cs file in the console application.

## **Build and Run:**

- Build the solution.
- Run the console application.

### **Que3. Conditional statement in c#**

In C#, conditional statements are used to control the flow of execution based on certain conditions. The primary conditional statements in C# are if, else if, else, and the switch statement. Here are examples of each:

#### **1. if Statement:**

The if statement allows you to execute a block of code if a specified condition is true.

```
Int number = 10;
```

```
if (number > 0)
{
    Console.WriteLine("The number is positive.");
}
```

#### **2. if-else Statement:**

The if-else statement allows you to execute one block of code if the condition is true and another block if the condition is false.

```
int number = -5;
```

```
if (number > 0)
{
    Console.WriteLine("The number is positive.");
}
else
{
    Console.WriteLine("The number is non-positive.");
}
```

#### **3. else if Statement:**

The else if statement is used to test multiple conditions. It is executed if the preceding if or else if conditions are false and its own condition is true.

```
int number = 0;
```

```
if (number > 0)
```

```
{
```

```
    Console.WriteLine("The number is positive.");
```

```
}
```

```
else if (number < 0)
```

```
{
```

```
    Console.WriteLine("The number is negative.");
```

```
}
```

```
else
```

```
{
```

```
    Console.WriteLine("The number is zero.");
```

```
}
```

```
{
```

```
    Console.WriteLine("The number is positive.");
```

```
}
```

#### **4. switch Statement:**

The switch statement is used when you have multiple possible conditions to test. It provides a concise way to compare a variable against multiple values.

```
int dayOfWeek = 3;
```

```
switch (dayOfWeek)
```

```
{
```

```
case 1:
Console.WriteLine("Monday");

    break;

case 2:
Console.WriteLine("Tuesday");

    break;

case 3:
Console.WriteLine("Wednesday");

    break;

// ... other cases ...

default:
Console.WriteLine("Invalid day");

    break;
}
```

#### **Que4. Loops statements in C#**

##### **1. for Loop:**

The for loop is used when you know in advance how many times you want to execute a block of code.

```
for (int i = 0; i < 5; i++)
{
    Console.WriteLine($"Iteration {i + 1}");
}
```

##### **2. while Loop:**

The while loop is used when you want to repeat a block of code as long as a specified condition is true.

```
int count = 0;
```

```
while (count < 5)
{
    Console.WriteLine($"Iteration {count + 1}");
    count++;
}
```

### **3. do-while Loop:**

The do-while loop is similar to the while loop, but it ensures that the block of code is executed at least once before checking the condition

```
int count = 0;
```

```
do
{
    Console.WriteLine($"Iteration {count + 1}");
    count++;
} while (count < 5);
```

### **4. foreach Loop:**

The foreach loop is used to iterate over elements in an array or a collection.

```
int[] numbers = { 1, 2, 3, 4, 5 };
```

```
foreach (int number in numbers)
{
    Console.WriteLine($"Number: {number}");
}
```

## **Que5. Classes in C#**

### **1. Declaring a Class:**

You declare a class using the class keyword, followed by the class name. The body of the class contains fields, properties, methods, and other members.



```

public class MyClass
{
    // Fields

    private int myField;


    // Properties

    public int MyProperty
    {
        get { return myField; }

        set { myField = value; }
    }


    // Methods

    public void MyMethod()
    {
        Console.WriteLine("Executing MyMethod");
    }
}

```

## 2. Creating Objects (Instances):

Once a class is defined, you can create objects (instances) of that class. Objects represent real entities based on the class blueprint.

```
MyClass myObject = new MyClass();
```

## 3. Accessing Members:

You can access the members (fields, properties, methods) of a class using the dot notation.

```
myObject.MyProperty = 42;
```

```
int value = myObject.MyProperty;
```

```
myObject.MyMethod();
```

#### **4. Constructors:**

A constructor is a special method used to initialize the object when it is created. It has the same name as the class.

```
public class MyClass  
{  
    private int myField;  
  
    // Constructor  
    public MyClass(int initialValue)  
    {  
        myField = initialValue;  
    }  
}
```

#### **5. Inheritance:**

C# supports inheritance, allowing one class to inherit the members of another. The : base keyword is used to specify the base class.

```
public class DerivedClass :MyBaseClass  
{  
    // Additional members for the derived class  
}
```

#### **6. Encapsulation:**

Classes provide encapsulation, which means bundling the data (fields) and methods that operate on the data within a single unit. Access modifiers like public, private, and protected control the visibility of class members.

#### **7. Example Usage:**

```
class Program  
{
```

```

static void Main()
{
    // Create an instance of MyClass
    MyClassmyObject = new MyClass(10);

    // Access members
    myObject.MyMethod();

    Console.WriteLine($"Value: {myObject.MyProperty}");
}
}

```

## **Que6. Delegates in C#**

### **1. Delegate Declaration:**

To declare a delegate, you specify the method signature it can reference. The delegate keyword is used for this purpose.

```
public delegate void MyDelegate(string message);
```

In the above example, MyDelegate is a delegate that can reference methods taking a string parameter and returning void.

### **2. Using Delegates:**

Once a delegate is declared, you can create an instance of it and associate it with methods that match its signature.

```

public class MyClass
{
    public void Method1(string message)
    {
        Console.WriteLine($"Method1: {message}");
    }
}

```

```

    public void Method2(string message)
    {
        Console.WriteLine($"Method2: {message}");
    }
}

```

```

class Program

```

```

{
    static void Main()
    {
        MyClassmyObject = new MyClass();

```

```

        // Create delegate instances and associate with methods

```

```

        MyDelegate delegate1 = myObject.Method1;

```

```

        MyDelegate delegate2 = myObject.Method2;

```

```

        // Invoke delegates

```

```

        delegate1("Hello");

```

```

        delegate2("World");

```

```

    }
}

```

### 3. Multicast Delegates:

Delegates can be combined to create a multicast delegate, which can invoke multiple methods.

```

MyDelegatemultiDelegate = delegate1 + delegate2;

```

```

multiDelegate("Combined invocation");

```

#### 4. Built-in Delegates:

C# provides several built-in generic delegate types in the System namespace, such as Action and Func. These are widely used and eliminate the need to define custom delegates for many scenarios.

```
Action<string>actionDelegate = myObject.Method1;  
  
actionDelegate("Using Action");
```

```
Func<int, int, int>addDelegate = (a, b) => a + b;  
  
int result = addDelegate(3, 5);  
  
Console.WriteLine($"Result: {result}");
```

#### 5. Events:

Delegates are commonly used to implement events, which provide a mechanism for one object to notify other objects when a specific event occurs.

```
public class Publisher  
{  
    public event MyDelegateMyEvent;  
  
    public void RaiseEvent(string message)  
    {  
        MyEvent?.Invoke(message);  
    }  
}  
  
class Program  
{  
    static void Main()  
    {  
        Publisher publisher = new Publisher();
```

```
MyClass subscriber = new MyClass();

    // Subscribe to the event
publisher.MyEvent += subscriber.Method1;

    // Raise the event
publisher.RaiseEvent("Event triggered");
}
}
```

### **Que7. File Handling in C#**

#### **1. Reading from a File:**

```
string filePath = "example.txt";

if (File.Exists(filePath))
{
    string[] lines = File.ReadAllLines(filePath);

    // Process lines
}
```

#### **2. Writing to a File:**

```
string filePath = "example.txt";

string[] lines = { "Line 1", "Line 2", "Line 3" };

File.WriteAllLines(filePath, lines);
```

#### **3. Appending to a File:**

```
string filePath = "example.txt";

string[] newLines = { "New Line 1", "New Line 2" };

File.AppendAllLines(filePath, newLines);
```

#### **4. Reading and Writing Binary Files:**

```
string filePath = "binaryfile.bin";
```

```
// Writing binary data
```

```
byte[] data = { 0x48, 0x65, 0x6C, 0x6C, 0x6F };
```

```
File.WriteAllBytes(filePath, data);
```

```
// Reading binary data
```

```
byte[] buffer = File.ReadAllBytes(filePath);
```

```
// Process binary data
```

### **Que8. Create a windows form app individual studio with C#**

#### **Steps to Create a Windows Forms Application:**

##### **1.Open Visual Studio:**

- Launch Visual Studio.

##### **2.Create a New Project:**

- Go to "File" -> "New" -> "Project..."
- In the "Create a new project" dialog, select "Windows Forms App (.NET Core)" or "Windows Forms App (.NET Framework)" based on your preference and system setup.
- Click "Next."

##### **3.Configure Project:**

- Enter a name for your project.
- Choose the location where you want to save the project.
- Set the solution name (optional).
- Choose the framework version (e.g., .NET Core 3.1 or .NET Framework 4.8).
- Click "Create."

##### **4.Design the Form:**

- Once the project is created, you'll see the default form (Form1.cs) in the designer.
- You can design your form by dragging and dropping controls from the Toolbox (View -> Toolbox) onto the form.
- Customize the properties of the controls using the Properties window.

##### **5.Add Code to the Form:**

- Double-click on a control to create an event handler.
- Add your C# code to handle events and perform actions.
- For example, you can add code to the button click event:

```
private void button1_Click(object sender, EventArgs e)

{

    MessageBox.Show("Hello, Windows Forms!");

}
```

## 6.Build and Run:

- Build your project by clicking on "Build" -> "Build Solution."
- Run your application by pressing F5 or clicking on the "Start Debugging" button.

That's it! You've created a simple Windows Forms application. You can further enhance your application by adding more controls, implementing additional functionality, and exploring features provided by the Windows Forms framework.

## Que9. Assembly in .net

### 1.Types of Assemblies:

**Single-File Assemblies (EXE):** Contains all the necessary information in a single executable file with the .exe extension.

**Multi-File Assemblies:** Comprises multiple files, including one main .exe file and accompanying .dll files containing additional code.

### 2.Components of an Assembly:

**Manifest:** Contains metadata about the assembly, such as version information, culture, public key token for strong naming, and a list of files that make up the assembly.

**MSIL (Microsoft Intermediate Language) Code:** The compiled code that is platform-independent and needs further compilation by the Just-In-Time (JIT) compiler at runtime.

### 3.Strong Naming:

Assemblies can be strongly named, which involves signing the assembly with a cryptographic key pair. Strong naming ensures uniqueness and integrity of the assembly.

### 4.Private and Shared Assemblies:

**Private Assemblies:** Used by a single application and stored in the application's directory.

**Shared Assemblies (Global Assembly Cache - GAC):** Accessible by multiple applications, allowing for code reuse. Shared assemblies are stored in the GAC.



## **5.Versioning:**

Assemblies support versioning, allowing multiple versions of the same assembly to coexist. This helps in managing updates and ensuring backward compatibility.

## **6.Deployment:**

Assemblies can be deployed along with the application or shared among multiple applications. Deployment options include XCOPY deployment, ClickOnce deployment, and deployment through installers.

## **7.References:**

Assemblies can reference other assemblies, and this reference information is stored in the manifest. References help in resolving dependencies between different components.

## **8.Reflection:**

.NET provides reflection, which allows runtime inspection of the metadata and types within an assembly. This enables dynamic loading and invocation of types.

## **Global Assembly Cache (GAC):**

The GAC is a machine-wide repository for shared assemblies. Shared assemblies in the GAC are accessible to multiple applications.

## **Que10. Window based calculator in C#**

### **1.Create a new Windows Forms Application:**

- Open Visual Studio.
- Create a new project: "File" -> "New" -> "Project..."
- Choose "Windows Forms App (.NET Core)" or "Windows Forms App (.NET Framework)" based on your preference.
- Name your project and click "Create."

### **2.Design the Calculator Form:**

- In the Solution Explorer, double-click on "Form1.cs" to open the designer.
- Drag and drop buttons for digits (0-9), operators (+, -, \*, /), equals (=), clear (C), and a TextBox for display.

### **3.Add Code to Handle Button Clicks:**

Double-click on each button to create event handlers. Add the following code to handle button clicks and perform calculations:

```
using System;
```

```
using System.Windows.Forms;
```

```
namespace CalculatorApp
{
    public partial class Form1 : Form
    {
        private string currentInput = "";
        private double currentValue = 0;
        private char currentOperator;

        public Form1()
        {
            InitializeComponent();
        }

        private void DigitButton_Click(object sender, EventArgs e)
        {
            Button button = (Button)sender;
            currentInput += button.Text;
            DisplayText(currentInput);
        }

        private void OperatorButton_Click(object sender, EventArgs e)
        {
            Button button = (Button)sender;
            if (!string.IsNullOrEmpty(currentInput))
            {
```

```
currentValue = double.Parse(currentInput);
```

```
currentInput = "";
```

```
currentOperator = button.Text[0];
```

```
    }
```

```
}
```

```
private void EqualsButton_Click(object sender, EventArgs e)
```

```
{
```

```
    if (!string.IsNullOrEmpty(currentInput))
```

```
    {
```

```
        double secondValue = double.Parse(currentInput);
```

```
        double result = PerformCalculation(currentValue, secondValue, currentOperator);
```

```
DisplayText(result.ToString());
```

```
currentInput = result.ToString();
```

```
    }
```

```
}
```

```
private void ClearButton_Click(object sender, EventArgs e)
```

```
{
```

```
currentInput = "";
```

```
currentValue = 0;
```

```
currentOperator = '\0';
```

```
DisplayText("");
```

```
}
```

```
private double PerformCalculation(double firstValue, double secondValue, char op)
```

```

{
    switch (op)
    {
        case '+':
            return firstValue + secondValue;

        case '-':
            return firstValue - secondValue;

        case '*':
            return firstValue * secondValue;

        case '/':
            return secondValue != 0 ? firstValue / secondValue : double.NaN;

        default:
            return double.NaN;
    }
}

private void DisplayText(string text)
{
    textBox1.Text = text;
}
}
}

```

#### 4. Build and Run:

- Build your project by clicking on "Build" -> "Build Solution."
- Run your application by pressing F5 or clicking on the "Start Debugging" button.