1.Demonstrate a Console Application program to validate Email-ID and username with 3 to 16 characters length and also can have all special characters using Regular Expressions.

Regular Expression in C#:
A Regular Expression (Regex) in C# is used to define patterns for matching strings.
-It is implemented through the System.Text.RegularExpressions namespace.
-The Regex class provides methods like:
IsMatch() → checks if a string matches the pattern.
Match() → returns the first match found.
Matches() → returns all matches in a string.
Replace() → replaces matched text with another string.
Split() → splits a string based on a pattern.

Code:

```
using System;
using System.Text.RegularExpressions;

class Program
{
   static void Main(string[] args)
   {
      string emailPattern =
@"^[^@\s]+@[^@\s]+\.[^@\s]+$";
      string usernamePattern =
@"^[\w\d!@#$%^&*()\-
_=+{};:,<.>]{3,16}$";
      Console.Write("Enter your Email ID: ");
      string email = Console.ReadLine();
      if (Regex.IsMatch(email,
emailPattern))
         Console.WriteLine("Valid Email ID.");
      else
         Console.WriteLine("Invalid Email
ID.");
      Console.Write("Enter your Username:
");
      string username = Console.ReadLine();
      if (Regex.IsMatch(username,
usernamePattern))
         Console.WriteLine("Valid
Username.");
      else
         Console.WriteLine("Invalid
Username. Username must be 3 to 16
characters long and can include special
characters.");
   }
}
```

String Manipulation with the String Builder and String Classes: Demonstrate some basic string manipulation using methods of both StringBuilder and String classes.

```
using System;
using System.Text;

class Program
{
   static void Main(string[] args)
   {
      Console.WriteLine("String Class
Manipulation:");
```

2. Working with callbacks and delegates in C#: Demonstrate the use of delegates, callbacks, and synchronous and asynchronous method invocation.

Delegates are type-safe function pointers in C# used to reference methods dynamically.
Singlecast delegates point to a single method and invoke only that one when called.
Multicast delegates can reference multiple methods using + or += operators.
When invoked, multicast delegates execute all referenced methods sequentially.

```
using System;
using System.Threading;
class Program
{
   public delegate void ProcessDelegate(int
number);
   static void Main(string[] args)
   {
      // Synchronous delegate invocation
      Console.WriteLine("Synchronous Delegate
Invocation:");
      ProcessDelegate processDelegate = new
ProcessDelegate(ProcessNumber);
      processDelegate(10);
      // Using a delegate as a callback
      Console.WriteLine("\nUsing Delegate as a
Callback:");
      ExecuteWithCallback(20, ProcessNumber);
      // Asynchronous delegate invocation
      Console.WriteLine("\nAsynchronous Delegate
Invocation:");
      ProcessDelegate asyncProcessDelegate = new
ProcessDelegate(ProcessNumber);
      IAsyncResult asyncResult =
asyncProcessDelegate.BeginInvoke(30, null, null);
      // Do other work while the delegate executes
asynchronously
      Console.WriteLine("Doing other work in the
main thread...");
      // Wait for asynchronous operation to
complete
     asyncProcessDelegate.EndInvoke(asyncResult);
      Console.WriteLine("\nProgram has finished
execution.");
   }
   // Method that matches the delegate signature
   public static void ProcessNumber(int number)
   {
      Console.WriteLine($"Processing number:
{number}");
      Thread.Sleep(2000); // Simulate time-
consuming work
      Console.WriteLine($"Finished processing
number: {number}");
   }

   // Method that takes a callback delegate as a
parameter
   public static void ExecuteWithCallback(int
number, ProcessDelegate callback)
   {
      Console.WriteLine("Starting execution...");
      callback(number);
      Console.WriteLine("Finished execution.");
   }
}
```

3. Working with Interface Inheritance: Demonstrate the interface inheritance using explicit interface Implementation

In C#, interface inheritance allows oneinterface to inherit from one or more other interfaces. A class that implements such an interface must provide implementations for all members from the base interfaces as well. When a class implements multiple interfaces that contain methods with the same name, explicit interface implementation is used to avoid ambiguity. In this method, the interface name is prefixed to the method while implementing it.
Explicit Interface Implementation allows a class to implement interface members so they are only accessible through the interface, not the class object.
It is used to avoid naming conflicts when multiple interfaces have members with the same name.
Syntax: ReturnType InterfaceName.MethodName() { }
Access requires casting the object to the interface type.

Code:

```
using System;
namespace InterfaceInheritanceExample
{
   public interface IBaseInterface
   {
      void BaseMethod();
   }

   public interface IDerivedInterface :
IBaseInterface
   {
      void DerivedMethod();
   }

   public class MyClass : IDerivedInterface
   {
      void IBaseInterface.BaseMethod()
      {
         Console.WriteLine("BaseMethod
implementation from IBaseInterface.");
      }

      void IDerivedInterface.DerivedMethod()
      {
         Console.WriteLine("DerivedMethod
implementation from IDerivedInterface.");
      }
      public void ExecuteMethods()
      {
         IBaseInterface baseInterface = this;
         IDerivedInterface derivedInterface = this;
         baseInterface.BaseMethod();
         derivedInterface.DerivedMethod();
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         MyClass myClass = new MyClass();
         myClass.ExecuteMethods();
      }
   }
}
```

```
        string str = "Hello, World!";
        Console.WriteLine("Original String: " +
str);
        string concatenatedStr = str + "
Welcome to string manipulation.";
        Console.WriteLine("After
Concatenation: " + concatenatedStr);
        string substring = str.Substring(7, 5);
        Console.WriteLine("Substring (7, 5): "
+ substring);

        string replacedStr =
str.Replace("World", "C#");
        Console.WriteLine("After Replace: " +
replacedStr);
        string upperStr = str.ToUpper();
        string lowerStr = str.ToLower();
        Console.WriteLine("ToUpper: " +
upperStr);
        Console.WriteLine("ToLower: " +
lowerStr);
        string strWithSpaces = "   Hello,
World!   ";
        string trimmedStr =
strWithSpaces.Trim();
        Console.WriteLine("Trimmed String: '"
+ trimmedStr + "'");
        int length = str.Length;
        Console.WriteLine("Length of String: "
+ length);

        Console.WriteLine("\nStringBuilder
Class Manipulation:");
        StringBuilder sb = new
StringBuilder("Hello, World!");
        sb.Append(" Welcome to string
manipulation.");
        Console.WriteLine("After Append: " +
sb.ToString());
        sb.Insert(7, "beautiful ");
        Console.WriteLine("After Insert: " +
sb.ToString());
        sb.Remove(7, 10);
        Console.WriteLine("After Remove: " +
sb.ToString());
        sb.Replace("World", "C#");
        Console.WriteLine("After Replace: " +
sb.ToString());
        int sbLength = sb.Length;
        Console.WriteLine("Length of
StringBuilder: " + sbLength);
        sb.Clear();
        Console.WriteLine("After Clear: " +
sb.ToString());
    }
}
```

4. Working with Inheritance: Employee Management System Tokyo Company wants to register the new employee to their data.

.....

```
using System;

namespace EmployeeManagementSystem
{
    public class Person
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Gender { get; set; }
        public DateTime DOB { get; set; }

        public Person(string firstName, string lastName,
string gender, DateTime dob)
        {
            FirstName = firstName;
            LastName = lastName;
            Gender = gender;
            DOB = dob;
        }
    }

    public class Employee : Person
    {
        public string Department { get; set; }
        public string Location { get; set; }
        public int EmpSequence { get; set; }
        public int YearsOfExperience { get; set; }

        public Employee(string firstName, string lastName,
string gender, DateTime dob,
            string department, string location, int
empSequence, int yearsOfExperience)
            : base(firstName, lastName, gender, dob)
        {
            Department = department;
            Location = location;
            EmpSequence = empSequence;
            YearsOfExperience = yearsOfExperience;
        }

        public string GenerateEmailID()
        {
            return Registration.UserID + "@Tokyo.com";
        }

        public double CalculateSalary()
        {
            double netCompensation = YearsOfExperience *
100000;
            double hra = netCompensation * 0.10;
            return netCompensation + hra;
        }
    }

    public class Registration : Employee
    {
        public static int EmployeeID;
        public static string UserID;

        public Registration(string firstName, string lastName,
string gender, DateTime dob,
            string department, string location, int
empSequence, int yearsOfExperience)
            : base(firstName, lastName, gender, dob,
department, location, empSequence, yearsOfExperience)
        {
            GenerateEmployeeID();
            GenerateUserID();
        }

        public void GenerateEmployeeID()
        {
```

5.Using Reflection in C#: Demonstrate how to gather information on various types included in any assembly by using the System.Reflection namespace and some main.NET base classes.

Reflection in C# is the ability of a program to inspect and interact with its own metadata at runtime.
It allows you to obtain information about assemblies, types, methods, properties, and fields dynamically.
Reflection enables creating objects, invoking methods, and accessing members without knowing them at compile time.
It is commonly used in frameworks, serialization, testing, and dynamic type discovery.

For reflection in C#, the required classes are mainly in the System.Reflection namespace. Key classes include:
1 .Assembly – Represents a .NET assembly and allows loading and inspecting assemblies.
2. Type – Represents type metadata (classes, interfaces, structs, enums) and provides information about members.
3 .MethodInfo – Represents a method and allows invocation and inspection of parameters and return types.
4. PropertyInfo – Represents a property and allows getting/setting values dynamically.
5.FieldInfo – Represents a field and allows reading/writing values dynamically.
6.ConstructorInfo – Represents a constructor and allows dynamic object creation.
7.ParameterInfo – Represents information about method parameters.

```
using System;
using System.Reflection;

namespace ReflectionDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            Assembly assembly =
Assembly.GetExecutingAssembly();
            Console.WriteLine($"Assembly Full Name:
{assembly.FullName}\n");

            Type[] types = assembly.GetTypes();
            foreach (Type type in types)
            {
                Console.WriteLine($"Type:
{type.FullName}");

                MethodInfo[] methods =
type.GetMethods();
                foreach (MethodInfo method in
methods)
                {
                    Console.WriteLine($" Method:
{method.Name}");

                    ParameterInfo[] parameters =
method.GetParameters();
                    foreach (ParameterInfo parameter in
parameters)
                    {
                        Console.WriteLine($"  Parameter:
{parameter.Name} of Type:
{parameter.ParameterType}");
                    }
```

6: Working with Assemblies:

An assembly is a compiled code library used by .NET applications. It contains metadata, intermediate language (IL) code, and resources. Assemblies are the building blocks of .NET applications and provide versioning, security, and deployment features.

Private Assembly:

A private assembly is used by a single application and stored in the application's folder. It does not require strong naming or GAC installation. Accessible only by the application that references it.

Public Assembly:

A public assembly is strong-named, stored in the Global Assembly Cache (GAC), and can be shared across multiple applications. It allows versioning and centralized management.

Shared Assembly:

A shared assembly is a public assembly that is strong-named and stored in the Global Assembly Cache (GAC). It can be used by multiple applications on the same machine, enabling code reuse and centralized versioning.

1. Demonstrate a console application by creating a Private Assembly and use it in different applications.

Step 1: Create a Private Assembly (Class Library)

1. Open Visual Studio as administrator.
2. Create a new **Class Library (.NET Framework)** project. **Name:** MyMathLibrary
3. Add the following class:

```
using System;

namespace MyMathLibrary
{
   public class Calc
   {
      public int add(int a, int b)
      {
         return a + b;
      }
   }
}
```

4. Build the solution.
5. Check the .dll file location (usually in bin\Debug or bin\Release).

Step 2: Create a Console Application to Use the Private Assembly

1. Create a new **Console Application** project named MathApp.
2. In **Solution Explorer**, right-click on **References** in the MathApp project → **Add Reference** → **Browse** → select MyMathLibrary.dll.
3. In Program.cs, write the following code:

```
using System;
using MyMathLibrary;

namespace MathApp
{
      EmployeeID = 10000 + EmpSequence;
    }

    public void GenerateUserID()
    {
      UserID = FirstName.Substring(0, 2).ToUpper() + EmployeeID;
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      Registration emp1 = new Registration("Rajesh", "Sharma", "Male", new DateTime(1990, 5, 12),
          "IT", "Tokyo", 101, 5);

      Console.WriteLine($"Employee ID: {Registration.EmployeeID}");
      Console.WriteLine($"User ID: {Registration.UserID}");
      Console.WriteLine($"Email ID: {emp1.GenerateEmailID()}");
      Console.WriteLine($"Salary: {emp1.CalculateSalary()}");
    }
  }
}
```

```
        Console.WriteLine($" Return Type: {method.ReturnType}");
      }

      PropertyInfo[] properties = type.GetProperties();
      foreach (PropertyInfo property in properties)
      {
        Console.WriteLine($" Property: {property.Name} of Type: {property.PropertyType}");
      }

      FieldInfo[] fields = type.GetFields();
      foreach (FieldInfo field in fields)
      {
        Console.WriteLine($" Field: {field.Name} of Type: {field.FieldType}");
      }

      Console.WriteLine();
    }
  }

  public class SampleClass
  {
    public int SampleField;
    public string SampleProperty { get; set; }

    public void SampleMethod1()
    {
      Console.WriteLine("SampleMethod1 called.");
    }

    public int SampleMethod2(int param1, string param2)
    {
      Console.WriteLine($"SampleMethod2 called with parameters: {param1}, {param2}");
      return param1 + param2.Length;
    }
  }
}
```

```
    internal class Program
    {
        static void Main(string[] args)
        {
            Calc c1 = new Calc();
            int x = c1.add(5, 10);
            Console.WriteLine(x);
        }
    }
}
```

2. Demonstrate how to Create a Public assembly and store it in GAC and use it in all applications.

Step 1: Create a Public Assembly (Class Library)

1. Open Visual Studio as administrator.
2. Create a new Class Library (.NET Framework) project. Name: MyPublicMathLibrary
3. Add the following class:

```
using System;

namespace MyPublicMathLibrary
{
    public class Calc
    {
        public int multiply(int a, int b)
        {
            return a * b;
        }
    }
}
```

4.Sign the assembly to make it strong-named (required for GAC):
Right-click on the project → Properties → Signing → check Sign the assembly → select New… → give a key name (e.g., MyKey.snk).
5.Build the solution.
6. The .dll file will be generated (usually in bin\Debug or bin\Release).

Step 2: Install the Assembly in GAC

1. Open Developer Command Prompt for Visual Studio as administrator.
2. Use the GacUtil tool to install the assembly:

Step 3: Use the Public Assembly in Any Application

1. Create a Console Application (e.g., PublicMathApp).
2. Add reference to the assembly from GAC:

Right-click References → Add Reference → Browse → select the GAC path or use the .dll location.

3. In Program.cs, write the following code:

```
using System;
using MyPublicMathLibrary;

namespace PublicMathApp
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Calc c1 = new Calc();
            int result = c1.multiply(5, 10);
            Console.WriteLine(result);
        }
    }
}
```

8.Using the System.NetWebClient to Retrieve or Upload Data: Demonstrate how to create windows form that can use HTTP to download and save a resource from a specified URI, upload a resource to a specified URI, or read and write data through a stream object.

Step 1: Create a Windows Forms Project

1. Open Visual Studio → Create a new project → Windows Forms App (.NET Framework) → Name it WebClientDemo.
2. Open Form1 in the designer view.

Step 2: Add Controls Using Drag-and-Drop
From the Toolbox, drag the following controls onto the form:

Control	Name	Properties / Notes
Label	lblURL	Text = "Enter URL:"
TextBox	txtURL	Width = 300
Button	btnDownload	Text = "Download"
Button	btnUpload	Text = "Upload"
TextBox	txtData	Multiline = True, Width = 300, Height = 100
Button	btnReadStream	Text = "Read Stream"
Button	btnWriteStream	Text = "Write Stream"

Step 3: Add Event Handlers

1. Double-click each Button in the designer → Visual Studio will create click event handlers in Form1.cs.
2. Add the following code inside Form1.cs inside the corresponding button methods:

```
using System;
using System.IO;
using System.Net;
using System.Text;
using System.Windows.Forms;

namespace WebClientDemo
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnDownload_Click(object sender, EventArgs e)
        {
            using (WebClient wc = new WebClient())
            {
                try
                {
                    string url = txtURL.Text;
                    string savePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Desktop), "downloaded_file");
                    wc.DownloadFile(url, savePath);
                    MessageBox.Show($"File downloaded to: {savePath}");
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error: " + ex.Message);
                }
            }
        }

        private void btnUpload_Click(object sender, EventArgs e)
        {
            using (WebClient wc = new WebClient())
            {
```

10. Working with LINQ:

Definition:
LINQ is a component of .NET that allows querying collections, databases, XML, and other data sources in a type-safe, readable, and consistent way using C# syntax .

Purpose:

--Provides a unified approach to query different data sources.
--Eliminates the need for separate query languages like SQL or XQuery for different sources.

Create the Database Table:
In SQL Server, run the following SQL to create a table:

```
CREATE TABLE Employees (
EmpNo INT PRIMARY KEY,
EmpName VARCHAR(50),
EmpSal DECIMAL(10, 2),
EmpJob VARCHAR(50),
EmpDeptNo INT
);
```

Step 1: Create Project
1. Open Visual Studio 2022
2. Select ASP.NET Web Application (.NET Framework)
3. Name it: EmployeeInfoApp
4. Choose Empty template and check Web Forms.

Step 2: Add Web Form
1. Right-click your project → Add → Web Form
o Name: EmployeeDisplay.aspx

Step 3: Create Database Connection
2. In Solution Explorer → Add → New Item → Data → LINQ to SQL Classes
o Name it: EmployeeData.dbml
3. This opens a LINQ to SQL Designer.
4. From Server Explorer → Data Connections, add connection
SERVER NAME =.\SQLEXPRESS
Connection need to be established with back end.
Open new query and Create the table Employee with required columns
5. drag the Employee table to the designer.
o It automatically creates a class named Employee.

Step 4: Design the Web Form (Front-End)
Open EmployeeDisplay.aspx, and add this code inside the <form> tag:

```
<asp:GridView ID="GridView1" runat="server"
AutoGenerateColumns="True"
</asp:GridView>
```

Step 5: Backend Code (C#)
Open EmployeeDisplay.aspx.cs and write this:

```
using System;
using System.Linq;
using System.Configuration;
namespace EmployeeInfoApp
{
public partial class EmployeeDisplay :
System.Web.UI.Page
{
protected void Page_Load(object sender,
EventArgs e)
{
```

```
            try
            {
                string url = txtURL.Text;
                string filePath =
Path.Combine(Environment.GetFolderPath(Environme
nt.SpecialFolder.Desktop), "upload_file.txt");
                File.WriteAllText(filePath, "This is a test
upload.");
                byte[] response = wc.UploadFile(url, "POST",
filePath);
                MessageBox.Show("Upload response: " +
Encoding.UTF8.GetString(response));
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
        }
    }
    private void btnReadStream_Click(object sender,
EventArgs e)
    {
        using (WebClient wc = new WebClient())
        {
            try
            {
                string url = txtURL.Text;
                using (Stream stream = wc.OpenRead(url))
                using (StreamReader reader = new
StreamReader(stream))
                {
                    txtData.Text = reader.ReadToEnd();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
        }
    }
    private void btnWriteStream_Click(object sender,
EventArgs e)
    {
        using (WebClient wc = new WebClient())
        {
            try
            {
                string url = txtURL.Text;
                byte[] data =
Encoding.UTF8.GetBytes(txtData.Text);
                wc.UploadData(url, "PUT", data);
                MessageBox.Show("Data uploaded
successfully.");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
        }
    }
  }
}
```

Step 4: Run the Application

1. Press F5 to run the form.
2. Test each button:

Download → saves a file from URL to Desktop.
Upload → uploads a file to a server.
Read Stream → loads content from URL into TextBox.
Write Stream → sends TextBox content to a server.

```
if (!IsPostBack)
{
ShowEmployeeData();
}
}
private void ShowEmployeeData()
{
// Create data context
string connString =
ConfigurationManager.ConnectionStrings
["DataSourceConnectionString"].ConnectionString;
// Create data context
empDataContext db = new
empDataContext(connString);
// LINQ Query to select employee details
var empDetails = from emp in db.Employees
select emp;
// Bind to GridView
GridView1.DataSource = empDetails;
GridView1.DataBind();
} } }
```

Create 5 content pages and design it accordingly and use different Navigation controls to navigate between content pages.

1. Create Project
    1. Open Visual Studio → ASP.NET Web Application (.NET Framework)
    2. Name: SMS1 → Empty template → Check Web Forms
2. Add Master Page
    1. Right-click project → Add → Master Page → Site1.Master
    2. Drag controls:

Header: <h1>CVR COLLEGE OF ENGINEERING</h1>
Menu: ASP:Menu with links to all pages
ContentPlaceHolder: ID=ContentPlaceHolder1
Footer: © 2025 SMS. All rights reserved.

3. Add 5 Content Pages
Right-click project → Add → Web Form using Master Page → select Site1.Master:

Page Name	Purpose
Home.aspx	Dashboard / Welcome
AddStudent.aspx	Add new student
ViewStudent.aspx	Display students in GridView
UpdateStudent.aspx	Update student info
About.aspx	About system / college info

4. Add Navigation Controls on Master Page

```
<!-- Menu Control -->
<asp:Menu ID="Menu1" runat="server"
Orientation="Horizontal">
  <Items>
    <asp:MenuItem Text="Home"
NavigateUrl="~/Home.aspx" />
    <asp:MenuItem Text="Add Student"
NavigateUrl="~/AddStudent.aspx" />
    <asp:MenuItem Text="View Students"
NavigateUrl="~/ViewStudent.aspx" />
    <asp:MenuItem Text="Update Student"
NavigateUrl="~/UpdateStudent.aspx" />
    <asp:MenuItem Text="About"
NavigateUrl="~/About.aspx" />
  </Items>
</asp:Menu>

<!-- HyperLink Controls -->
<asp:HyperLink runat="server"
NavigateUrl="~/Home.aspx">Home</asp:HyperLink> |
<asp:HyperLink runat="server"
NavigateUrl="~/AddStudent.aspx">Add
Student</asp:HyperLink> |
<asp:HyperLink runat="server"
NavigateUrl="~/ViewStudent.aspx">View
Students</asp:HyperLink> |
<asp:HyperLink runat="server"
NavigateUrl="~/UpdateStudent.aspx">Update
Student</asp:HyperLink> |
<asp:HyperLink runat="server"
NavigateUrl="~/About.aspx">About</asp:HyperLink>

<!-- SiteMapPath -->
<asp:SiteMapPath ID="SiteMapPath1" runat="server" />

<!-- TreeView -->
<asp:TreeView ID="TreeView1" runat="server"
DataSourceID="SiteMapDataSource1" />
<asp:SiteMapDataSource ID="SiteMapDataSource1"
runat="server" ShowStartingNode="False" />
```

5. Design Content Pages (Drag-and-Drop)

Home.aspx

	Welcome to Student Management System AddStudent.aspx <asp:Label Text="Student Name:" /><asp:TextBox ID="txtName" runat="server" /> <asp:Label Text="Age:" /><asp:TextBox ID="txtAge" runat="server" /> <asp:Button ID="btnAdd" runat="server" Text="Add Student" /> ViewStudent.aspx All Students <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="True"></asp:GridView> UpdateStudent.aspx <asp:Label Text="Student ID:" /><asp:TextBox ID="txtStudentID" runat="server" /> <asp:Button ID="btnLoad" runat="server" Text="Load Details" /> About.aspx About SMS Information about the system and college. 6. Result Master Page: Consistent header, footer, and navigation. Navigation: Users can use Menu, HyperLink, TreeView, SiteMapPath, or buttons to move between pages. Content Pages: Display different information inside ContentPlaceHolder1.	