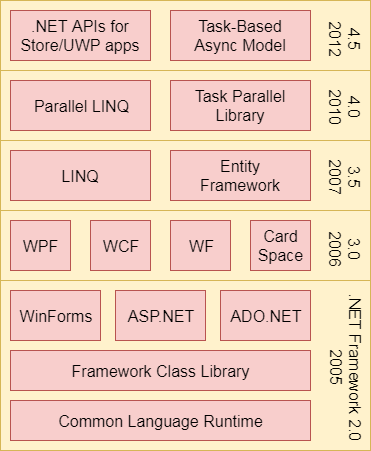
PREREQUISITES:

Introduction to .NET Framework and Installing Visual Studio.

## .NET Framework Overview

.NET is a software development framework developed by Microsoft, released in 2000. It supports the development of applications for Windows, web, and mobile platforms and offers a variety of functionalities such as security, memory management, networking, and type-safety. It includes **Framework Class Library (FCL)** and the **Common Language Runtime (CLR)**. With .NET, you can use more than 60 programming languages, including C#, F#, VB.NET, and more.



The .NET Framework is composed of four main components:

1. Common Language Runtime (CLR)
2. Framework Class Library (FCL),
3. Core Languages (WinForms, ASP.NET, and ADO.NET), and
4. Other Modules (WCF, WPF, WF, Card Space, LINQ, Entity Framework, Parallel LINQ, Task Parallel Library, etc.)

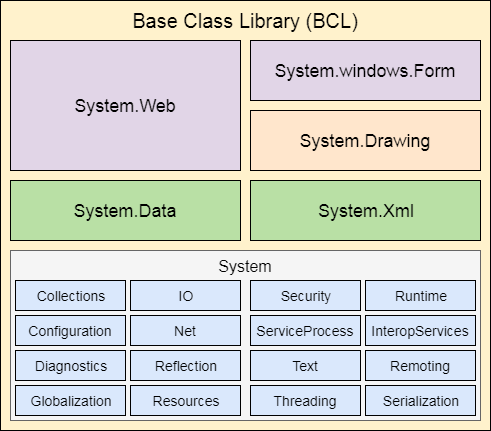
## Common Language Runtime (CLR):

* + Manages code execution, memory management, and garbage collection.
  + Acts as the interface between the framework and the operating system.
  + Provides services like exception handling, type-safety, and security.



## Framework Class Library (FCL):

* + A vast collection of reusable classes and libraries.
  + The **Base Class Library (BCL)** offers fundamental functions like data types, I/O operations, and collections.
  + Simplifies the development process by offering pre-built functionality.



## Core Languages:

* + **WinForms**: Simplifies building Windows desktop applications with a user-friendly interface.
  + **ASP.NET**: A framework to build web applications, websites, and services with seamless HTML, CSS, and JavaScript integration.
  + **ADO.NET**: Used to interact with databases, enabling data retrieval, manipulation, and storage.

## Other Modules:

* + Includes technologies like **WCF (Windows Communication Foundation)**, **WPF (Windows Presentation Foundation)**, **LINQ (Language Integrated Query)**, **Entity Framework**, and **Parallel LINQ** for advanced development capabilities.

## Installing Visual Studio

Visual Studio is the Integrated Development Environment (IDE) used to develop .NET applications.

## Download Visual Studio:

* + Visit the [Visual Studio Downloads](https://visualstudio.microsoft.com/downloads/) page.
  + Choose the edition you want to install (Community, Professional, Enterprise). The Community edition is free and suitable for most developers.

## Run the Visual Studio Installer:

* + Open the installer after downloading.
  + Choose the workloads that match your development needs (e.g., ".NET desktop development," "ASP.NET and web development," etc.).
  + Click **Install**.

## Start Visual Studio:

* + After installation, open Visual Studio.
  + Sign in with your Microsoft account if needed.

## Create a New Project:

* + Open Visual Studio and click on **Create a new project**.
  + Choose a project template such as **Console App (.NET Core)**, **ASP.NET Web Application**, or **WPF Application** depending on your needs.
  + Set the project name, location, and click **Create**.

## Experiment Number:

**Date of Experiment:**

## Aim:

Write a program to implement a simple Calculator using Windows Application.

## Description:

**Create a New Project**:

1. Open **Visual Studio**.
2. Click on **"Create a new project"**.
3. From the list of project templates, search for **"Windows Forms App"** (make sure to choose either

**.NET Core** or **.NET Framework** depending on your requirements).

1. Click **Next**, give your project a name, choose a location to save it, and click **Create**. **Open the Form Designer**:
2. Once the project is created, you'll see the **Solution Explorer** on the right side.
3. In the **Solution Explorer**, you'll find **Form1.cs** (the default form). Double-click on it to open the

## Windows Form Designer.

1. This opens the visual designer where you can drag and drop controls like buttons, text boxes, and labels onto the form.

## Design the Form:

In the Form Designer, you can visually design your UI by dragging elements from the **Toolbox** (which is usually on the left side of the screen) onto your form.

## Write Code for the Form:

To write code behind the form (e.g., handling button clicks or events), double-click on a control (e.g., a button). Visual Studio will open the code-behind file (Form1.cs), where you can add logic to handle user interaction.

## Run the Form:

Press **F5** or click the **Start** button (green arrow) to compile and run the Windows Form application. This will launch the form you've designed.

## Program:

namespace CalculatorApp

{

public partial class CalculatorForm : Form

{

public CalculatorForm()

{

InitializeComponent();

}

// Variables for calculation string operation;

int firstNumber; int secondNumber;

int calculationResult;

// Button click handlers for numeric buttons private void btn1\_Click(object sender, EventArgs e)

{

txtDisplay.Text += btn1.Text;

}

private void btn2\_Click(object sender, EventArgs e)

{

txtDisplay.Text += btn2.Text;

}

private void btn3\_Click(object sender, EventArgs e)

{

txtDisplay.Text += btn3.Text;

}

private void btn4\_Click(object sender, EventArgs e)

{

txtDisplay.Text += btn4.Text;

}

private void btn5\_Click(object sender, EventArgs e)

{

txtDisplay.Text += btn5.Text;

}

private void btn6\_Click(object sender, EventArgs e)

{

txtDisplay.Text += btn6.Text;

}

private void btn7\_Click(object sender, EventArgs e)

{

txtDisplay.Text += btn7.Text;

}

private void btn8\_Click(object sender, EventArgs e)

{

txtDisplay.Text += btn8.Text;

}

private void btn9\_Click(object sender, EventArgs e)

{

txtDisplay.Text += btn9.Text;

}

private void btn0\_Click(object sender, EventArgs e)

{

txtDisplay.Text += btn0.Text;

}

// Button click handlers for operations

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

{

operation = "+";

firstNumber = int.Parse(txtDisplay.Text); txtDisplay.Clear();

}

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

{

operation = "-";

firstNumber = int.Parse(txtDisplay.Text); txtDisplay.Clear();

}

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

{

operation = "\*";

firstNumber = int.Parse(txtDisplay.Text); txtDisplay.Clear();

}

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

{

operation = "/";

firstNumber = int.Parse(txtDisplay.Text); txtDisplay.Clear();

}

// Button click handler for the equals button

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

{

secondNumber = int.Parse(txtDisplay.Text); switch (operation)

{

case "+":

calculationResult = firstNumber + secondNumber; break;

case "-":

calculationResult = firstNumber - secondNumber; break;

case "\*":

calculationResult = firstNumber \* secondNumber; break;

case "/":

calculationResult = firstNumber / secondNumber; break;

}

txtDisplay.Text = calculationResult.ToString();

}

// Button click handler for clearing the input

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

{

txtDisplay.Clear(); firstNumber = 0;

secondNumber = 0;

calculationResult = 0;

}

// Form load event (if needed in the future)

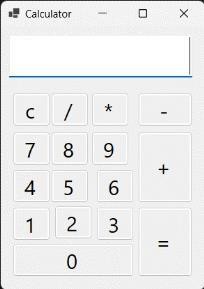
private void CalculatorForm\_Load(object sender, EventArgs e) { }

}

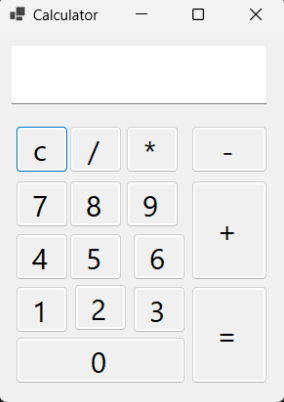
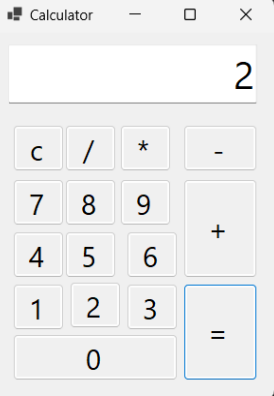
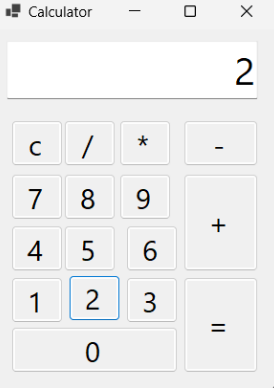
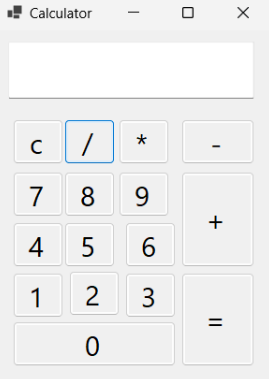
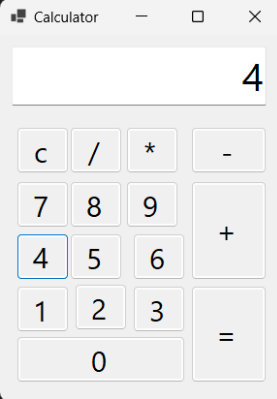
}

## Output:

This is the created form of the calculator.

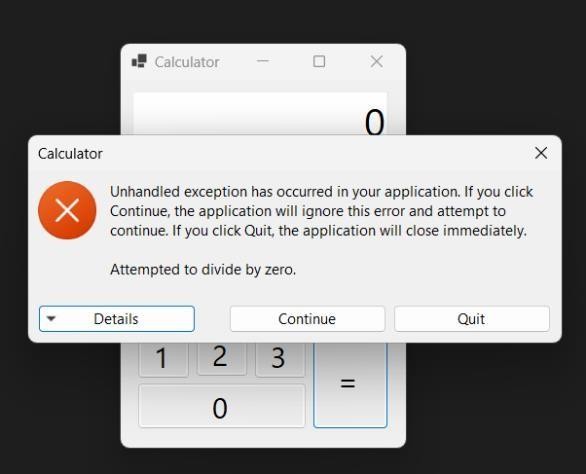
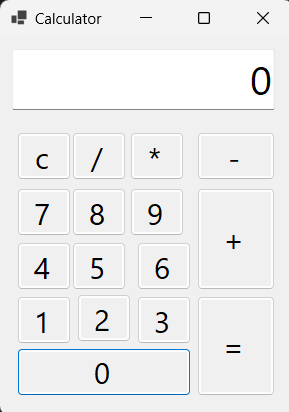
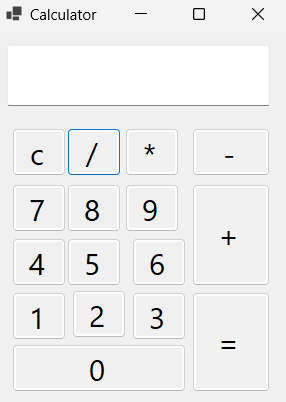
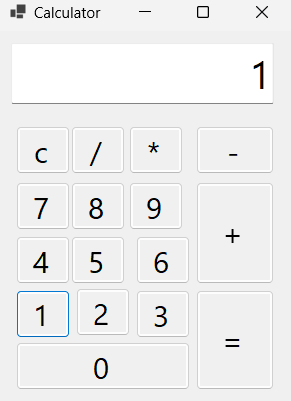


Implementing the basic operation **4 / 2 = 2**.



After clicking the button “C” which is clear, the input box will be cleared.

Implementing the basic operation **1 / 0 = infinite**



## Experiment Number:

**Date of Experiment:**

## Aim:

Write a program to implement a Student Information Form Submission.

## Description:

Step 1: Open Visual Studio 2022

Before you begin, ensure that Visual Studio 2022 is installed on your computer with the .NET desktop development workload. This is needed to create Windows Forms applications.

1. Open Visual Studio 2022.

Step 2: Create a New Windows Forms Project

1. Launch Visual Studio 2022.
2. Click on Create a new project from the start page.
3. In the Create a new project window:
   * In the search box, type Windows Forms App to filter templates.
   * Select Windows Forms App (.NET Framework).
   * Click Next.
4. Configure your project:
   * Project Name: Enter a name for your project (e.g., StudentInformationForm).
   * Location: Choose where to save the project on your computer.
   * Solution Name: This is the name of the overall solution; typically, this can be the same as the project name.
   * Framework: Ensure you select the correct .NET Framework version (usually, Visual Studio 2022 uses .NET 5.0 or higher).
   * Click Create.

Step 3: Understanding the Project Structure

Once your project is created, you’ll see the following structure:

* + Form1.cs: This is the main form of the project where you design the UI and write code for events (like button clicks).
  + Program.cs: The entry point of your application (contains the Main method).
  + Properties: Contains various project settings.
  + References: External libraries and dependencies.

Step 4: Designing the User Interface (UI)

4.1.1 Open the Designer

By default, Form1.cs [Design] is open. If it’s not, double-click Form1.cs in the Solution Explorer and switch to the Design view by clicking the [Design] tab.

* 1. Add Controls to the Form

You’ll now design the UI by adding controls like Labels, TextBoxes, ComboBoxes, RadioButtons, ListBoxes, and a Submit button.

To add controls:

* + - Go to the Toolbox on the left side of Visual Studio. If it's not visible, go to View > Toolbox.
    - Drag and drop the following controls onto the form, positioning them as required for the form layout.

Step-by-Step for Adding Controls

1. Labels: Drag a Label from the Toolbox onto the form. Use these labels to indicate what each field represents (e.g., "Student Name", "Roll Number", etc.).
   * Set the Text property of each label via the Properties window (usually found on the right) to reflect the name of the field.

Example:

* + Label for "Student Name": Set the Text property to "Student Name".

1. TextBoxes: Drag TextBox controls from the Toolbox. These will be used for input fields (e.g., for entering student names, roll numbers, etc.).
   * Example:
     + TextBox for "Student Name": Place it next to the corresponding label and set the Name property to something like stuname.
2. ComboBoxes: Combo boxes are used to offer users a drop-down list of options (e.g., Year, Semester).
   * Drag a ComboBox onto the form.
   * Use the Items property in the Properties window to add predefined values like:
     + For Year: Add items like "1st Year", "2nd Year", "3rd Year", "4th Year".
     + Set the Name property to something like stuyear.
3. RadioButtons: Use RadioButton controls to offer mutually exclusive choices (e.g., Gender selection).
   * Drag RadioButton controls from the Toolbox and position them together for options like Male, Female, and Other.
   * Set their Text properties accordingly and assign Name properties like stu\_gender\_male, stu\_gender\_female, and stu\_gender\_other.
4. ListBox: Drag a ListBox onto the form for multi-selection options (e.g., clubs that students can join).
   * Populate the list items via the Items property with options like "Science Club", "Drama Club", "Art Club", etc.
   * Set the Name property to clubs\_list.
5. Submit Button: Finally, drag a Button onto the form for submitting the data.
   * Set the Text property of the button to "Submit".
   * Set the Name property to something like submit\_btn. Step 5: Writing Code for Button Click Event

Now that the form is designed, let’s handle the Submit button’s functionality, which will check if the fields are filled and display the data or give an alert.

1. Double-click the Submit Button: Double-click on the Submit button on your form. This action automatically creates an event handler for the button click event in Form1.cs.
2. Add the Code: In the generated event handler, you can write the logic you want to implement. Step 6: Running the Project
3. Save your project.
4. Click the Start button (or press F5) to compile and run the application.
5. Test the form by filling out the fields and clicking the Submit button. If fields are empty, the application will notify the user; otherwise, it will display the submission details.

## Program:

namespace WinFormsApp2 {

public partial class Form1 : Form { string g;

public Form1() { InitializeComponent();

}private void Form1\_Load(object sender, EventArgs e){ } private void groupBox1\_Enter(object sender, EventArgs e){ } private void ClearForm() {

sturoll.Text = string.Empty; stuname.Text = string.Empty; stuyear.SelectedIndex = -1;

stusem.SelectedIndex = -1; fathername.Text = string.Empty; mothername.Text = string.Empty; stuphn.Text = string.Empty; stumail.Text = string.Empty; stublood.Text = string.Empty; stucgpa.Text = string.Empty; stubacklogs.Text = string.Empty; clubs\_list.ClearSelected(); stu\_gender\_male.Checked = false; stu\_gender\_female.Checked = false; stu\_gender\_other.Checked = false;

}

private void submit\_btn\_Click(object sender, EventArgs e) {

if (stu\_gender\_male.Checked == true) { g = "Male";

} else if (stu\_gender\_female.Checked == true) { g = "Female";

} else {

g = "Other";

}

bool isAnyEmpty = false;

foreach (Control control in this.Controls) {

if (control is System.Windows.Forms.TextBox) { if (string.IsNullOrEmpty(control.Text)) {

isAnyEmpty = true; break;

}

}else if (control is System.Windows.Forms.ComboBox) {

if (((System.Windows.Forms.ComboBox)control).SelectedIndex == -1) { isAnyEmpty = true;

break;

}

} else if (control is ListBox) {

if (((ListBox)control).SelectedIndex == -1) { isAnyEmpty = true;

break;

}

}

}if (!stu\_gender\_male.Checked && !stu\_gender\_female.Checked && !stu\_gender\_other.Checked) { isAnyEmpty = true;

} if (isAnyEmpty) {

MessageBox.Show("One or more fields are empty, fill it before submit");

}else{

MessageBox.Show("Submission Completed. \n" + "Student Name : " + stuname.Text + "\nStudent RollNo: " + sturoll.Text

+ "\nStudent Semester: " + stuyear.Text + "\nStudent Semester: " + stusem.Text + "\nFather Name: " + fathername.Text + "\nMother Name: " + mothername.Text + "\nStudent Phone: " + stuphn.Text + "\nStudent Email: " + stumail.Text + "\nStudent Blood Group: " + stublood.Text + "\nStudent Gender: " + g + "\nStudent CGPA: " + stucgpa.Text + "\nStudent Backlogs: " + stubacklogs.Text + "\nClubs Interested: " + clubs\_list.Text);

ClearForm();

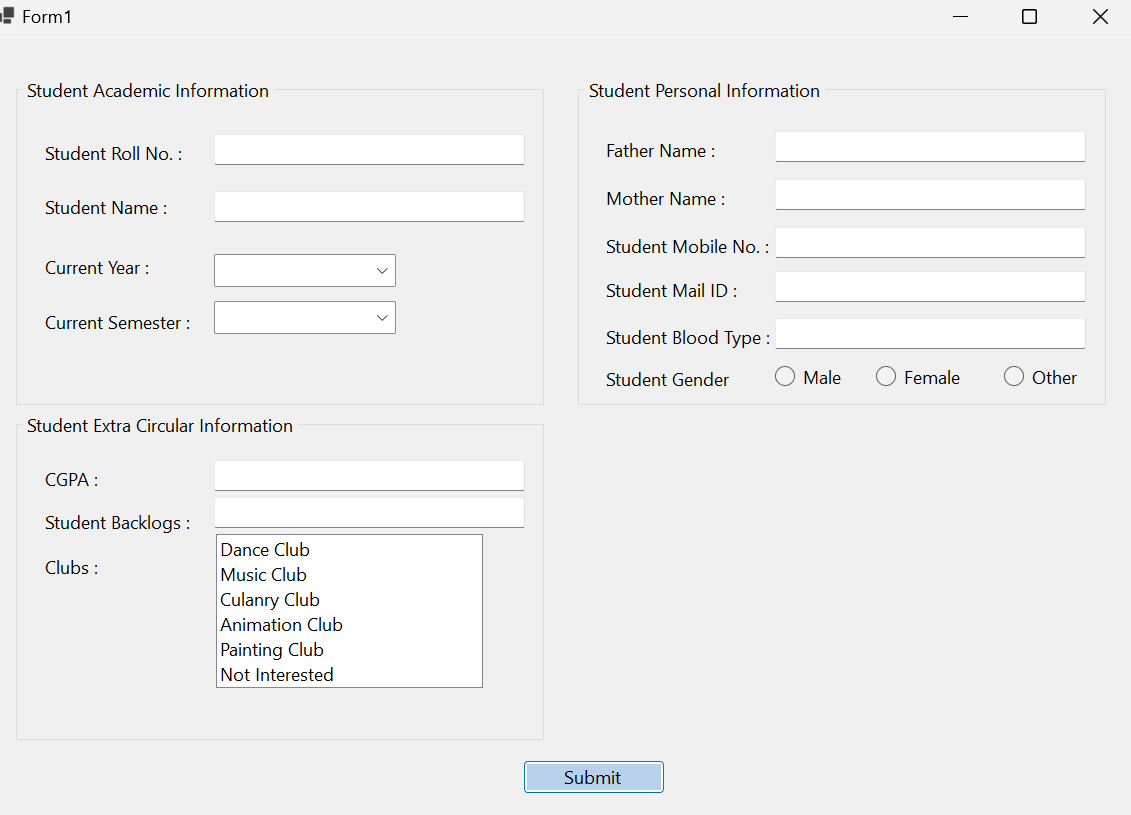
}

}

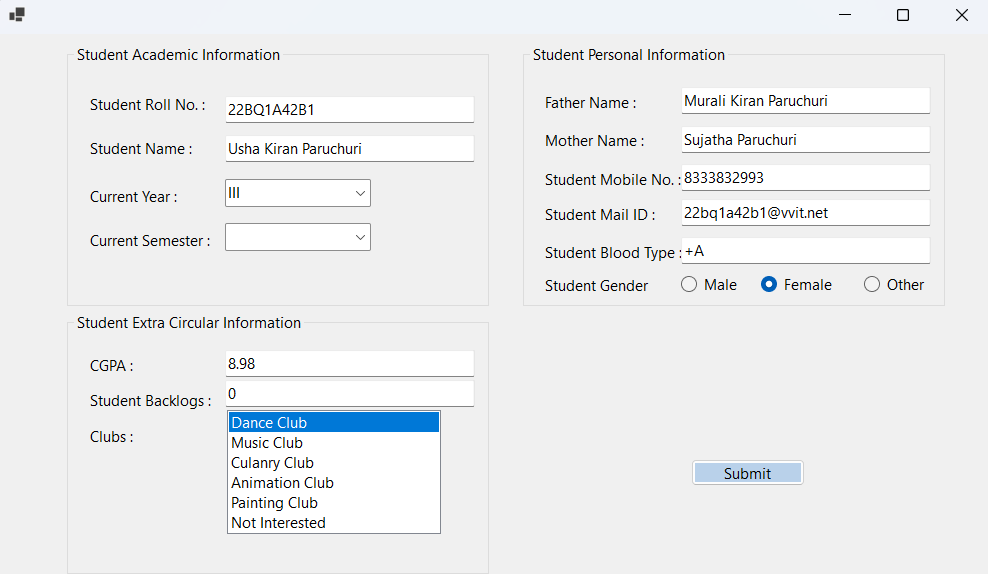
} }

## Output:

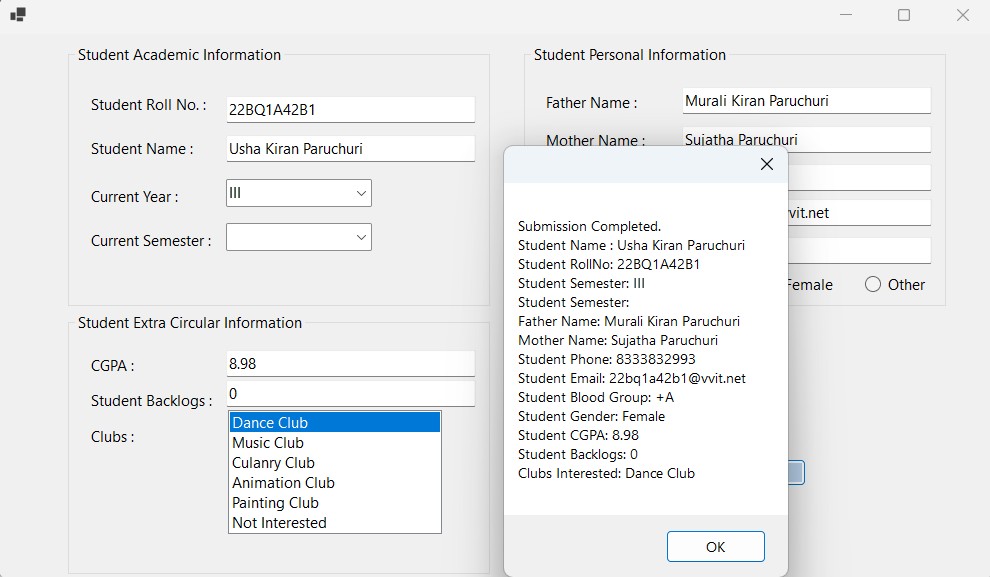
This is the created form of Student Information.



Fill the fields with the details and click the submit button.



After clicking submit button we got the message of submission with provided details.



## Experiment Number:

**Date of Experiment:**

## Aim:

Write a program to implement a simple MDI From.

## Description:

An **MDI (Multiple Document Interface)** form in .NET is a type of form that allows an application to host multiple child forms within a single parent form. It is a design pattern used to manage multiple documents or windows within a single application window, such as the way Microsoft Excel or Visual Studio organizes various files and projects in one interface.

## Key Concepts of MDI Forms in .NET:

1. **MDI Parent and MDI Child Forms:**
   * **MDI Parent:** This is the main container form that holds multiple child forms. It acts as the workspace for the entire application, and all child windows appear inside this parent window.
   * **MDI Child:** These are the individual forms or windows that open inside the parent form. The child forms are constrained to the boundaries of the parent form, and the user can interact with each of them independently.

In a typical MDI application, the user can open multiple child forms, and all of them will appear within the main window, providing a unified workspace for multiple tasks.

1. **Creating an MDI Application:** In .NET, you can create an MDI application by setting the parent form's property to designate it as an MDI container. Once the parent form is set, you can open child forms within it by assigning them to the parent. This establishes a relationship where the parent form manages the layout and behavior of all its child forms.
2. **MDI Layout Management:** MDI parent forms control how child windows are displayed. .NET offers options like:
   * **Cascade:** Child windows are arranged in an overlapping manner, with each window slightly offset.
   * **Tile Horizontal/Vertical:** Child windows are arranged either horizontally or vertically, making them all visible within the parent.
   * **Arrange Icons:** Minimizes child forms and arranges them neatly within the parent.

These layout options provide different ways to organize child forms, helping the user manage multiple windows in a clean, organized manner.

1. **Managing Child Forms:** In MDI applications, you often need to manage child forms programmatically. For instance, you may need to close all open child forms at once or ensure that only one instance of a particular child form can be opened at a time. This prevents confusion and clutter, as the application only allows a single instance of specific child forms (such as a calculator or settings window).
2. **Toolbars and Menus in MDI Forms:** Typically, MDI applications have toolbars and menus in the parent form, allowing users to interact with the child forms. For example, menus in the parent form may provide options for creating new child forms, closing them, or arranging them in different layouts. These menus and toolbars are part of the user interface of the parent form and control the behavior of the entire application.

## MDI Advantages:

* + **Centralized Management:** The parent form serves as a central workspace, managing all child forms in a unified interface.
  + **Better Organization:** It provides options for organizing multiple windows, such as tiling or cascading, making it easier to manage multiple tasks or documents.
  + **Single Application Window:** All child windows are contained within the main parent window, reducing desktop clutter and providing a cohesive user experience.

## MDI Disadvantages:

* + **Complex for Users:** Managing multiple windows inside one parent form can be confusing for users who are not familiar with this interface.
  + **Limited Usability on Modern UI:** MDI interfaces are less common in modern application design, which often favors single-document or tabbed interfaces for better user experience.

## Key Properties and Methods:

* **MDI Container Property:** The parent form is set as the MDI container, indicating that it can hold multiple child forms.
* **Child-Parent Relationship:** Each child form is associated with the parent form, and the parent controls the behavior and layout of the children.
* **Layout Options:** The parent form provides options like cascading, tiling, or arranging icons to manage the layout of the child windows.

Here we creates a Multiple Document Interface (MDI) application using C# and Windows Forms, where the main form (MDIParent1.cs) can host child forms such as a calculator (Form1.cs) and a student information form (Form2.cs). Each of these child forms is presented and managed within the main application window.

## Breakdown:

1. **MDIParent1.cs (Main MDI Parent Form)**
   * The main form (MDIParent1) is a container that can open and manage multiple child forms.
   * The MDI form includes several features like creating new forms, opening files, saving files, and managing layout (cascading, tiling, arranging icons, etc.).
   * **Child Forms:** The form manages different child forms (e.g., calculator and student info forms), ensuring that only one instance of each can be open at a time. If a child form is already open, the code activates the existing form rather than opening a new one.
   * **Form-specific logic:** When a child form is closed, it resets the reference to null so that it can be reopened later if necessary.
   * **Event Handling:** The form has event handlers for menu options, such as "New Window", "Open File", "Save File", and "Exit". These allow users to open new windows, save files, or close the application. Other functionalities include toggling the visibility of toolbars and status bars.
   * **MdiChild management:** The layout of child windows can be adjusted using options like cascading or tiling.

## Form1.cs (Calculator Form)

* + This is a basic calculator form with buttons for digits (0-9) and arithmetic operations (addition, subtraction, multiplication, and division).
  + **Button Click Logic:** Each digit button appends its value to the text box (tb1). Operation buttons store the current value in a variable (a), reset the text box, and set the operation type (msg).
  + **Result Calculation:** The equals button performs the operation by taking the second operand from the text box, performing the arithmetic based on the operation stored in msg, and then displaying the result.
  + **Clear Button:** The clear button resets the text box to an empty string.

## Form2.cs (Student Information Form)

* + This form collects and displays student information. It includes text boxes for entering details such as name, roll number, and other personal information.
  + **Radio Buttons:** It allows users to select the gender (Male, Female, Other).
  + **Combo Boxes:** Users can select the current year and semester from drop-down lists.
  + **Message Display:** When the "Submit" button is clicked, the form gathers all the input data and displays it in a message box, including student details, gender, clubs, and other information.

## Key Features and Concepts:

* **MDI (Multiple Document Interface):** The MDIParent1 class manages child forms within the main form, which is an MDI container. Multiple child forms can be opened inside this parent form, allowing for a flexible interface where different tasks can be performed simultaneously.
* **Event-driven Programming:** The application is highly event-driven, where actions like button clicks, form closes, or menu selections trigger specific behaviors, such as opening new forms or performing calculations.
* **Child Form Management:** The main form ensures only one instance of certain child forms (e.g., calculator or student info) can be opened at a time. This is managed through FormClosed event handlers that reset the references to the child forms upon their closure.
* **Basic Calculator Logic:** The calculator follows a simple two-step process where the user inputs two operands and an operator (e.g., addition or subtraction), then computes the result when the equal button is pressed.
* **Student Information Form:** This form provides a simple interface for entering student details, which are then displayed in a message box for review. The use of radio buttons, combo boxes, and text boxes provides a basic user interface for data input.

## Program:

**MDIParent1.cs**

using System;

using System.Collections.Generic; using System.ComponentModel; using System.Data;

using System.Drawing; using System.Linq; using System.Text;

using System.Threading.Tasks; using System.Windows.Forms; namespace MDI\_lab\_3\_{

public partial class MDIParent1 : Form { private int childFormNumber = 0; public MDIParent1() {

InitializeComponent();

}

private void ShowNewForm(object sender, EventArgs e) { Form childForm = new Form();

childForm.MdiParent = this;

childForm.Text = "Window " + childFormNumber++; childForm.Show();

}

private void OpenFile(object sender, EventArgs e) { OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Personal); openFileDialog.Filter = "Text Files (\*.txt)|\*.txt|All Files (\*.\*)|\*.\*";

if (openFileDialog.ShowDialog(this) == DialogResult.OK) { string FileName = openFileDialog.FileName;

}

}

private void SaveAsToolStripMenuItem\_Click(object sender, EventArgs e) { SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Personal); saveFileDialog.Filter = "Text Files (\*.txt)|\*.txt|All Files (\*.\*)|\*.\*";

if (saveFileDialog.ShowDialog(this) == DialogResult.OK) { string FileName = saveFileDialog.FileName;

}

}

private void ExitToolsStripMenuItem\_Click(object sender, EventArgs e) {

this.Close();

}

private void CutToolStripMenuItem\_Click(object sender, EventArgs e) { } private void CopyToolStripMenuItem\_Click(object sender, EventArgs e) { } private void PasteToolStripMenuItem\_Click(object sender, EventArgs e) { } private void ToolBarToolStripMenuItem\_Click(object sender, EventArgs e) {

toolStrip.Visible = toolBarToolStripMenuItem.Checked;

}

private void StatusBarToolStripMenuItem\_Click(object sender, EventArgs e) { statusStrip.Visible = statusBarToolStripMenuItem.Checked;

}

private void CascadeToolStripMenuItem\_Click(object sender, EventArgs e) { LayoutMdi(MdiLayout.Cascade);

}

private void TileVerticalToolStripMenuItem\_Click(object sender, EventArgs e) { LayoutMdi(MdiLayout.TileVertical);

}

private void TileHorizontalToolStripMenuItem\_Click(object sender, EventArgs e) { LayoutMdi(MdiLayout.TileHorizontal);

}

private void ArrangeIconsToolStripMenuItem\_Click(object sender, EventArgs e) { LayoutMdi(MdiLayout.ArrangeIcons);

}

private void CloseAllToolStripMenuItem\_Click(object sender, EventArgs e) { foreach (Form childForm in MdiChildren) {

childForm.Close();

}

}

Form1 f1;

private void calculatorToolStripMenuItem\_Click(object sender, EventArgs e) { if (f1 == null) {

f1 = new Form1(); f1.MdiParent = this;

f1.FormClosed += new FormClosedEventHandler(f1\_FormClosed); f1.Show();

} else {

f1.Activate();

}

}

f1 = null;

}

Form2 f2;

private void sTUInfoFormToolStripMenuItem\_Click(object sender, EventArgs e) { if (f2 == null) {

f2 = new Form2(); f2.MdiParent = this;

f2.FormClosed += new FormClosedEventHandler(f2\_FormClosed); f2.Show();

} else {

f2.Activate();

}

}

void f2\_FormClosed(object sender, FormClosedEventArgs e) { f2 = null;

}

Form3 f3;

private void form1ToolStripMenuItem\_Click(object sender, EventArgs e) { if (f3 == null) {

f3 = new Form3(); f3.MdiParent = this;

f3.FormClosed += new FormClosedEventHandler(f3\_FormClosed); f3.Show();

} else {

f3.Activate();

}

}

void f3\_FormClosed(object sender, FormClosedEventArgs e) { f3 = null;

}

Form4 f4;

private void form2ToolStripMenuItem\_Click(object sender, EventArgs e) { if (f4 == null) {

f4 = new Form4(); f4.MdiParent = this;

f4.FormClosed += new FormClosedEventHandler(f4\_FormClosed); f4.Show();

} else {

f4.Activate();

}

}

void f4\_FormClosed(object sender, FormClosedEventArgs e) {

f4 = null;

}

}

}

## Form1.cs (Calculator Form) :

namespace MDI\_lab\_3\_{

public partial class Form1 : Form{ int a = 0;

int b, c;

String msg;

public Form1(){ InitializeComponent();

}

private void btn1\_Click(object sender, EventArgs e) { tb1.Text += btn1.Text;

}

private void btn2\_Click(object sender, EventArgs e) { tb1.Text += btn2.Text;

}

private void btn3\_Click(object sender, EventArgs e) { tb1.Text += btn3.Text;

}

private void btn4\_Click(object sender, EventArgs e) { tb1.Text += btn4.Text;

}

private void btn5\_Click(object sender, EventArgs e) { tb1.Text += btn5.Text;

}

private void btn6\_Click(object sender, EventArgs e) { tb1.Text += btn6.Text;

}

private void btn7\_Click(object sender, EventArgs e) { tb1.Text += btn7.Text;

}

private void btn8\_Click(object sender, EventArgs e) { tb1.Text += btn8.Text;

}

private void btn9\_Click(object sender, EventArgs e) {

tb1.Text += btn9.Text;

}

private void btn0\_Click(object sender, EventArgs e) { tb1.Text += btn0.Text;

}

private void btn\_clear\_Click(object sender, EventArgs e) { tb1.Text = "";

}

private void btn\_add\_Click(object sender, EventArgs e) { msg = "+";

a = Convert.ToInt32(tb1.Text); tb1.Text = "";

}

private void btn\_sub\_Click(object sender, EventArgs e) { msg = "-";

a = Convert.ToInt32(tb1.Text); tb1.Text = "";

}

private void btn\_mul\_Click(object sender, EventArgs e) { msg = "\*";

a = Convert.ToInt32(tb1.Text); tb1.Text = "";

}

private void btn\_div\_Click(object sender, EventArgs e)

{

msg = "/";

a = Convert.ToInt32(tb1.Text); tb1.Text = "";

}

private void btn\_equal\_Click(object sender, EventArgs e) { if (msg == "+"){

b = Convert.ToInt32(tb1.Text); c = a + b;

tb1.Text = c.ToString();

}

if (msg == "-"){

b = Convert.ToInt32(tb1.Text); c = a - b;

tb1.Text = c.ToString();

}

if (msg == "\*"){

b = Convert.ToInt32(tb1.Text); c = a \* b;

tb1.Text = c.ToString();

}

if (msg == "/"){

b = Convert.ToInt32(tb1.Text); c = a / b;

tb1.Text = c.ToString();

}

}

private void Form1\_Load(object sender, EventArgs e) { }

}

}

## Form2.cs (Student Information Form) :

namespace MDI\_lab\_3\_ {

public partial class Form2 : Form { public Form2() {

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e) { String g;

if (radioButton1.Checked == true) { g = "Male";

} else if (radioButton2.Checked == true) { g = "Female";

} else {

g = "Other";

}

MessageBox.Show("Student Name: "+textBox2.Text+"\nStudent Roll No: "+textBox1.Text+"\nCurrent Year: "+comboBox1.Text+"\nCurrent Semester: "+comboBox2.Text+"\nFather Name: "+textBox7.Text+"\nMother Name: "+textBox6.Text+"\nStudent Mail: "+textBox5.Text+"\nStudent Blood Type: "+textBox4.Text+"\nStudent Gender: "+ g +"\nStudent CGPA: "+textBox8.Text+"\nStudent Backlogs: "+textBox9.Text+"\nClubs: "+listBox1.SelectedItem.ToString());

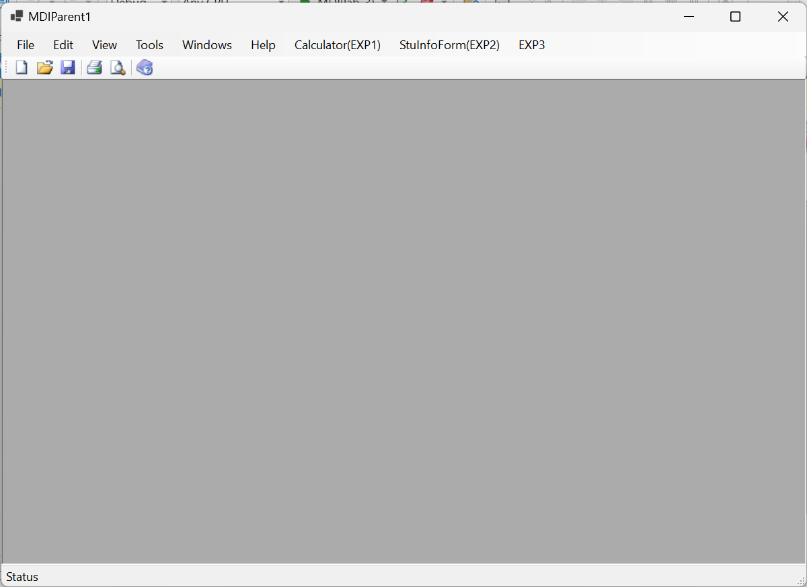
}

}

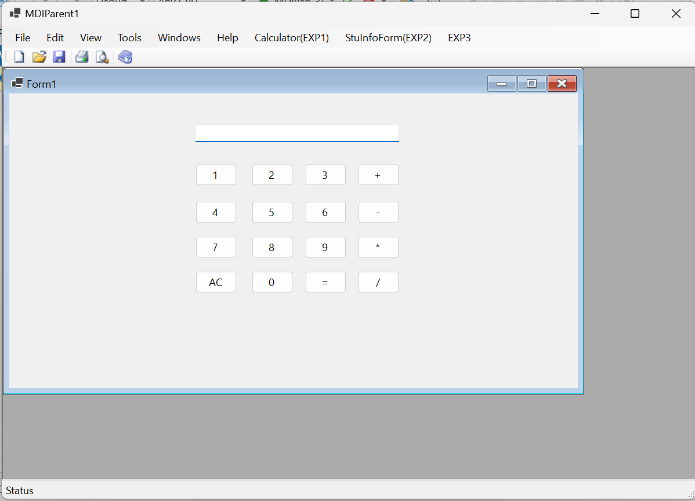
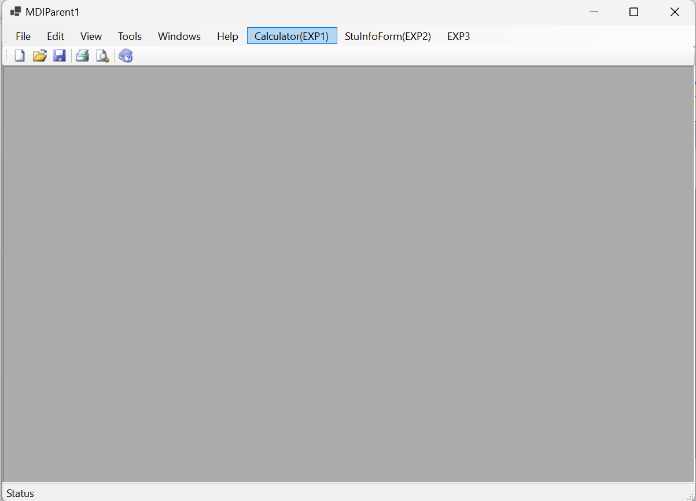
}

## Output:

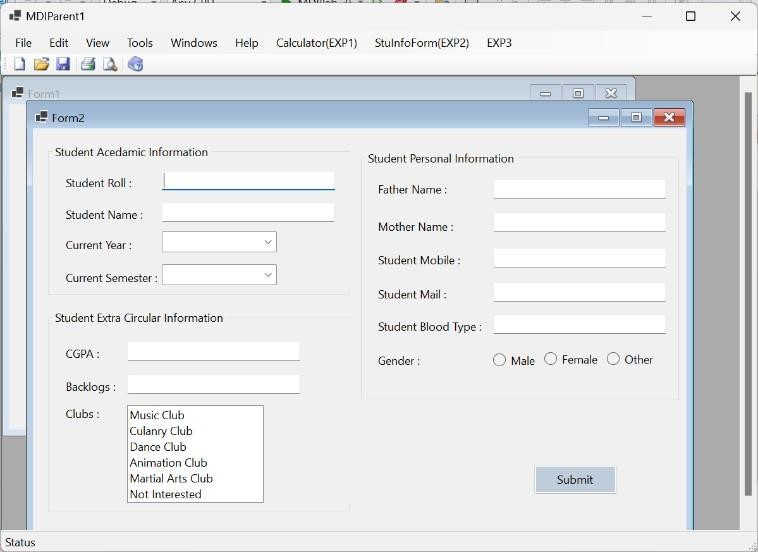
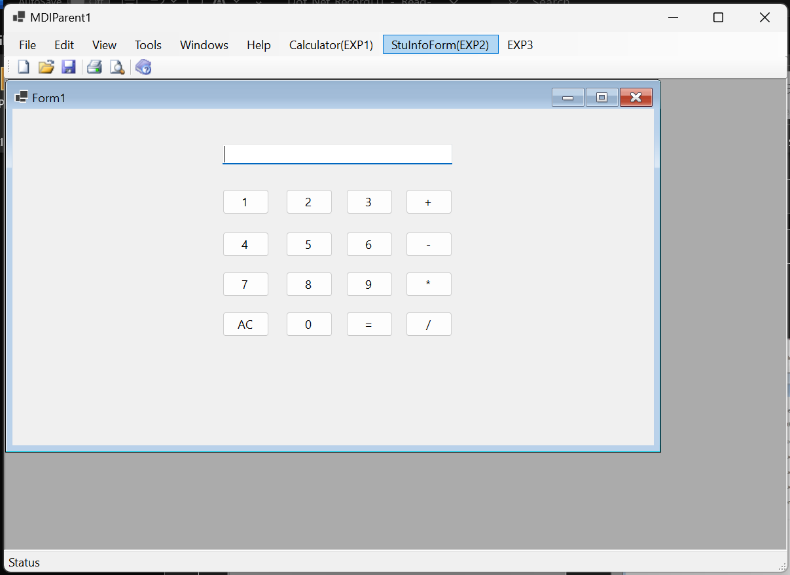
This is the created MDI Parent form.



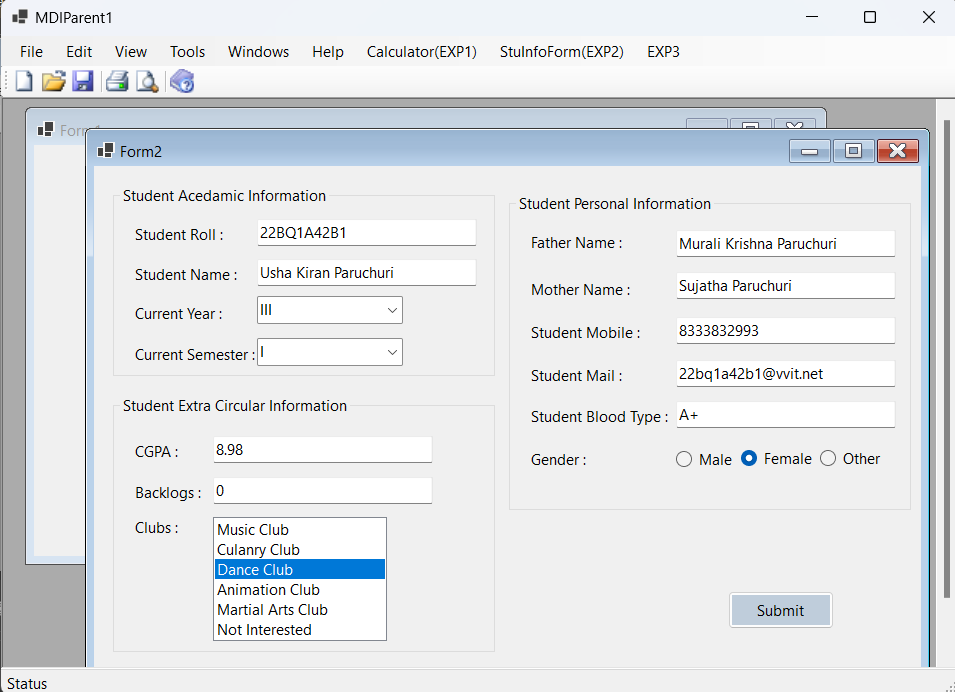
On clicking the Calculator it will open the calculator form

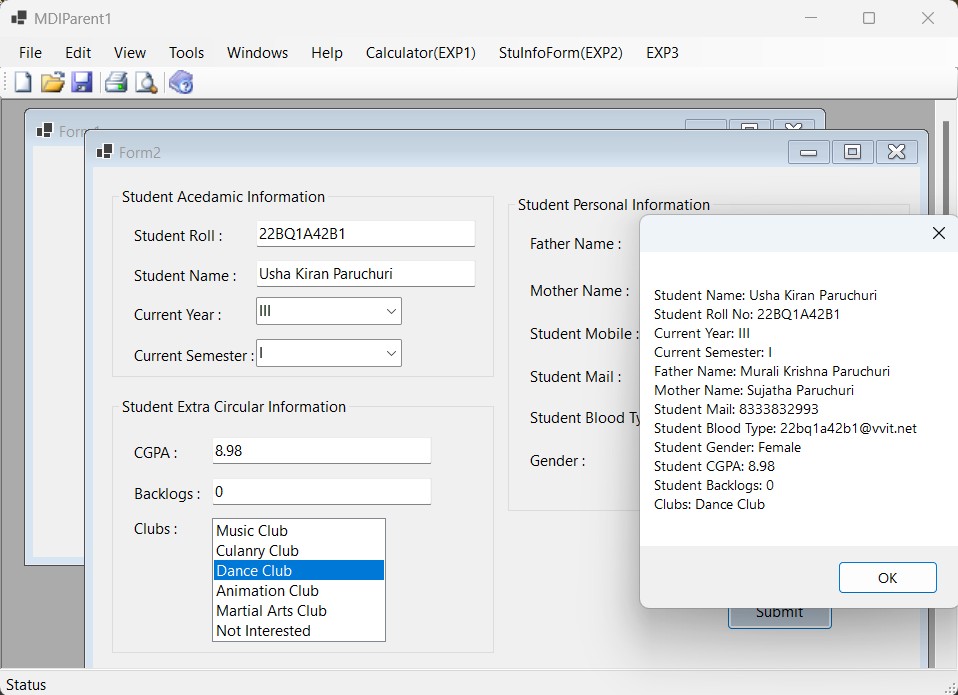


On clicking the StuInfoForm it will open the student information submission form.

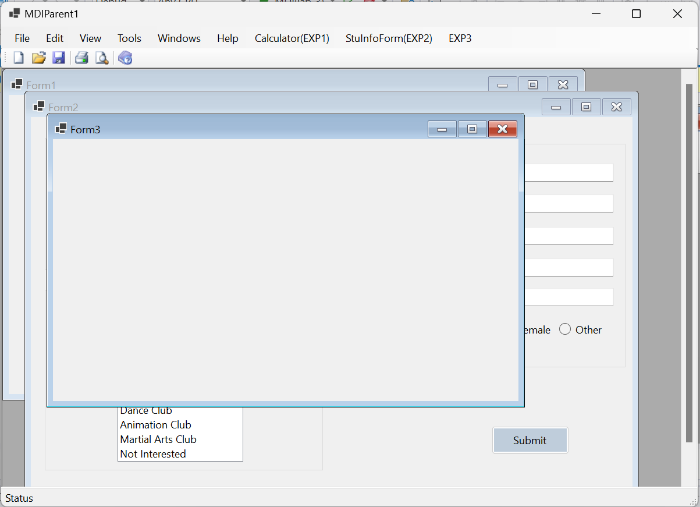
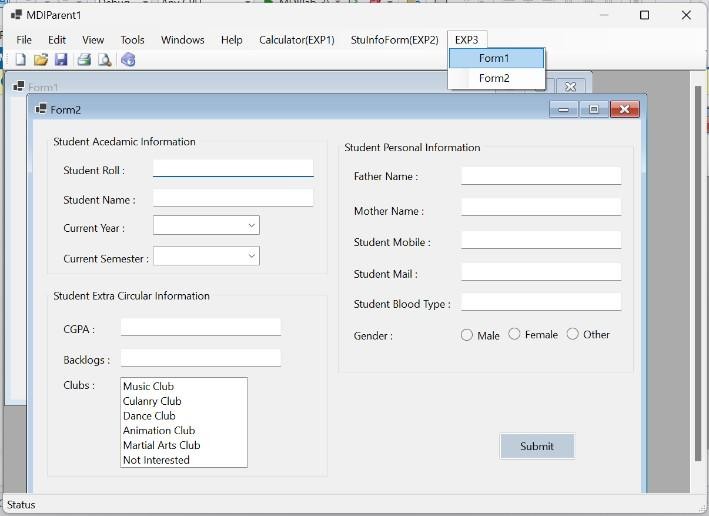


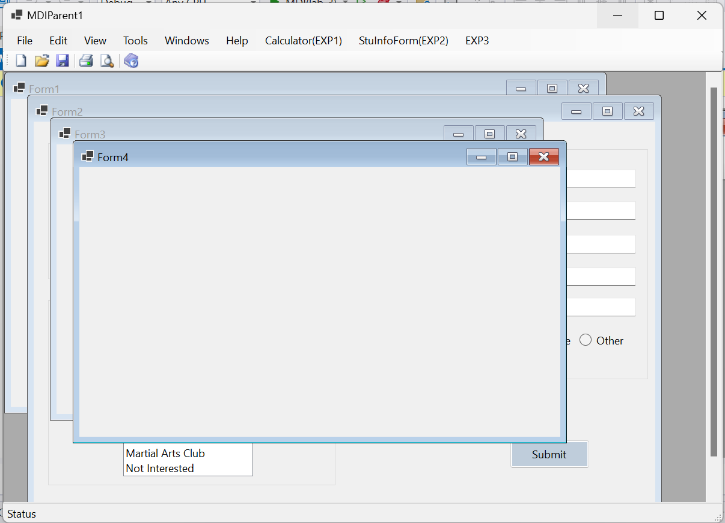
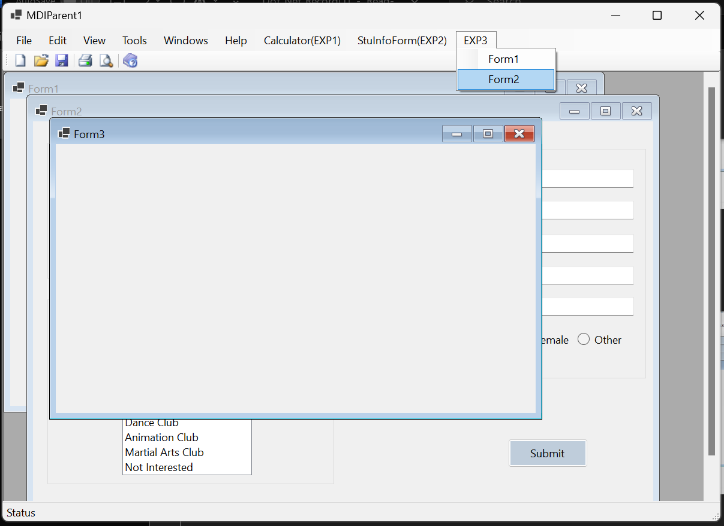
By filling the information in the given fields3, by submitting it will display the information that we have provided.



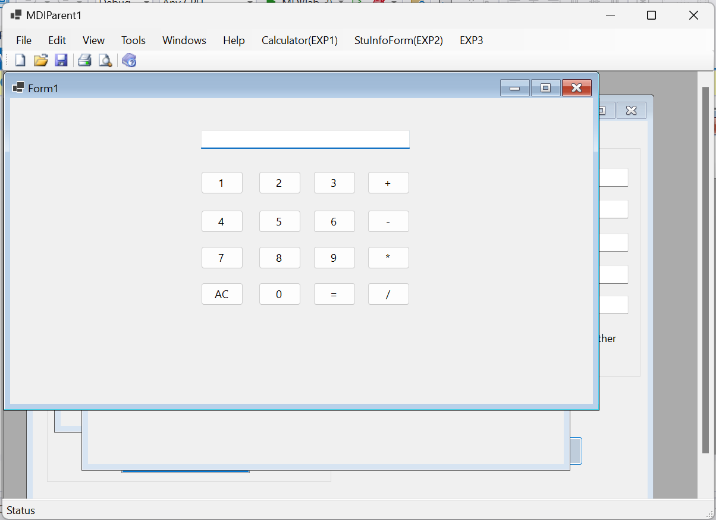
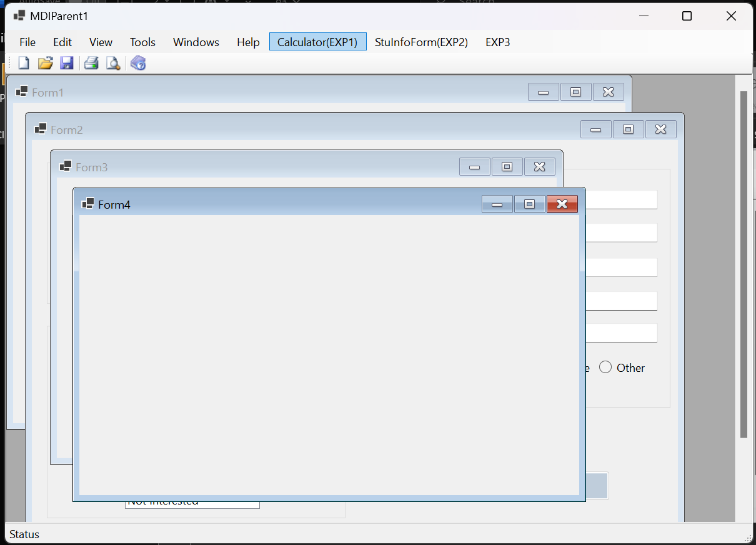


In the third field, we have the multiple form as an option , by clicking on each one, it will open each form.





By clicking the already opened form, it will notopen another one, it will pop the already existing one to the top.



## Experiment Number:

**Date of Experiment:**

## Aim:

Write a program to implement student Information form submission with database connectivity.

**Description:**

# Namespaces Used:

System.Windows.Forms: This is the core library for creating Windows Forms applications. System.Data.SqlClient: This is used for SQL operations and database connectivity.

System.Drawing: Used for working with graphics, but it's not specifically used in this form.

1. Form1 Class: This class is the main form where users will input their data. It inherits from Form and includes various controls like text boxes, radio buttons, combo boxes, and list boxes for data input.
2. Controls on the Form: The form contains several controls:
   * Textboxes (e.g., sturoll, stuname, fathername, etc.) to collect input such as student roll number, name, parent details, etc.
   * ComboBoxes (e.g., stuyear, stusem) for dropdown selection of values like year and semester.
   * Radio Buttons (stu\_gender\_male, stu\_gender\_female, stu\_gender\_other) for gender selection.
   * ListBox (clubs\_list) to select from a list of clubs.

# Functionality Breakdown

1. Gender Selection: The gender is determined by checking which radio button (male, female, other) is selected. Based on the checked button, the variable g is assigned a value (Male, Female, or Other).
2. Form Validation: The code iterates through all the controls on the form (this.Controls) to check if any of the required fields are left empty.
   * Textboxes: If a TextBox is empty (control.Text is empty or null), isAnyEmpty is set to true.
   * ComboBoxes: If no item is selected in a ComboBox (SelectedIndex == -1), isAnyEmpty becomes true.
   * ListBox: If no item is selected in the ListBox, isAnyEmpty becomes true.
   * Gender: If none of the gender radio buttons is checked, isAnyEmpty is also set to true.

If any fields are empty, a MessageBox pops up to notify the user that they must fill in all fields before submitting.

1. SQL Database Insertion: If all fields are filled in correctly, the data is inserted into the SQL database using the SqlConnection and SqlCommand classes.
   * Connection String (cs): It defines the SQL Server connection parameters.
   * Insert Query (insertQuery): This query is used to insert student information into the STUDENTINFORMATION table in the database.
   * Parameters: The SqlCommand object uses parameters like @STU\_ROLL, @STU\_NAME, etc., to insert the corresponding values from the form controls into the database.
2. Form Reset (ClearForm): After successfully submitting the form, the ClearForm method is called, which

resets all the fields and controls on the form back to their default (empty) state.

* + Textboxes are cleared by setting Text to string.Empty.
  + ComboBoxes are reset by setting SelectedIndex to -1.
  + ListBox is reset using ClearSelected().
  + Radio Buttons (for gender) are all unchecked.

# Database Connection:

Here are the detailed steps for creating a database, adding a table, and writing columns in Microsoft SQL Server using SQL Server Management Studio (SSMS):

Step 1: Open SQL Server Management Studio (SSMS)

1. Launch SQL Server Management Studio (SSMS) from your start menu or desktop.
2. In the Connect to Server window, select the server type (Database Engine), enter your Server Name, and provide the appropriate authentication (Windows Authentication or SQL Server Authentication).
3. Click Connect to connect to your SQL Server instance. Step 2: Create a New Database
4. In the Object Explorer (left pane), right-click on the Databases node.
5. Select New Database....
6. In the New Database window:
   * Enter the database name (e.g., STUDENTINFORMATIONDB).
   * Keep the default settings for now (unless you need specific configurations like file locations or size).
7. Click OK to create the new database. Step 3: Create a New Table
8. Expand the newly created database in the Object Explorer by clicking the + next to Databases and then your database (STUDENTINFORMATIONDB).
9. Right-click on the Tables node and select New Table....
10. The table designer will open, where you can define the columns for your table. Step 4: Define Columns

In the table designer, you'll see rows where you can input column definitions. Fill in the fields for each column as follows:

1. Column Name: This is the name of the column.
2. Data Type: Select the appropriate data type (e.g., INT, NVARCHAR, FLOAT, etc.).
3. Allow Nulls: Check this box if the column is allowed to have NULL values, or leave it unchecked to make the field mandatory.

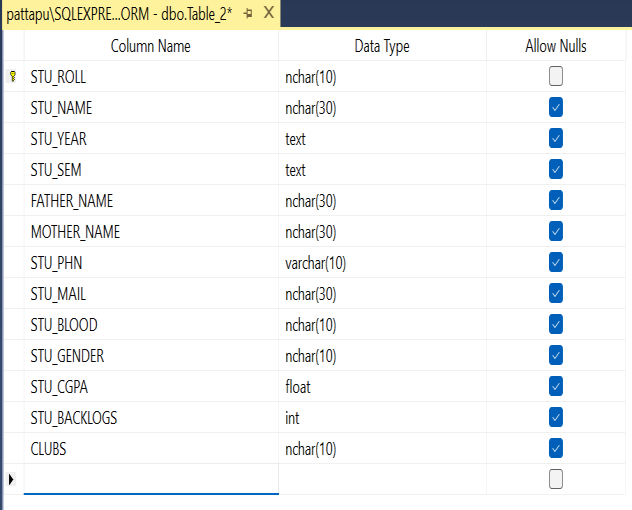
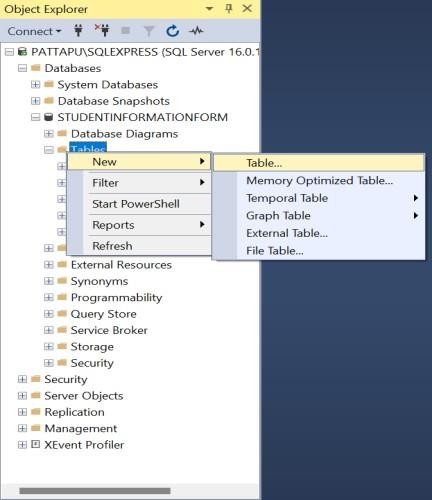
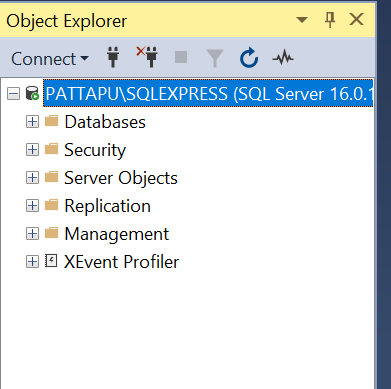
Define Columns for STUDENTINFORMATION Table:

1. STU\_ROLL: Set NCHAR(10), Primary Key (Do not allow nulls).
2. STU\_NAME: Set NCHAR(30), Allow Nulls.
3. STU\_YEAR: Set TEXT, Allow Nulls.
4. STU\_SEM: Set TEXT, Allow Nulls.
5. FATHER\_NAME: Set NCHAR(30), Allow Nulls.
6. MOTHER\_NAME: Set NCHAR(30), Allow Nulls.
7. STU\_PHN: Set VARCHAR(10), Allow Nulls.
8. STU\_MAIL: Set NCHAR(30), Allow Nulls.
9. STU\_BLOOD: Set NCHAR(10), Allow Nulls.
10. STU\_GENDER: Set NCHAR(10), Allow Nulls.
11. STU\_CGPA: Set FLOAT, Allow Nulls.
12. STU\_BACKLOGS: Set INT, Allow Nulls.
13. CLUBS: Set NCHAR(10), Allow Nulls.

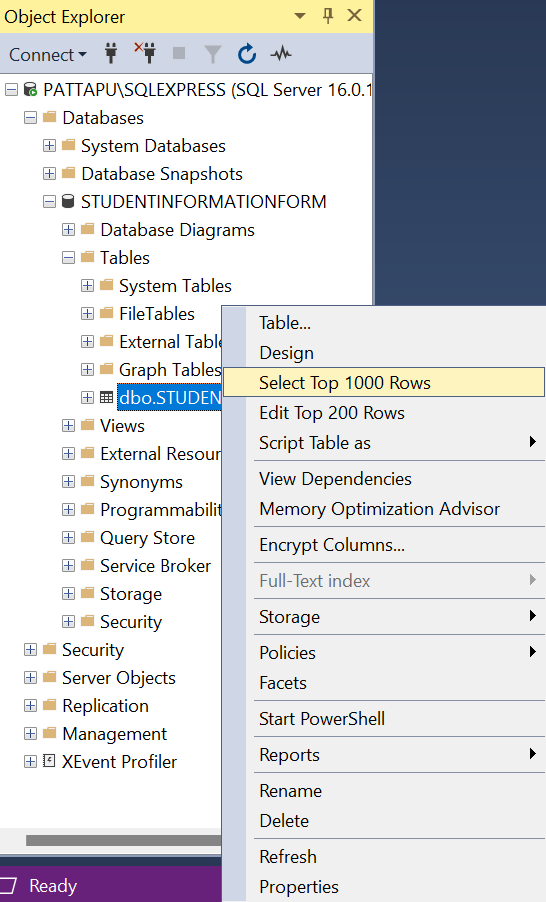
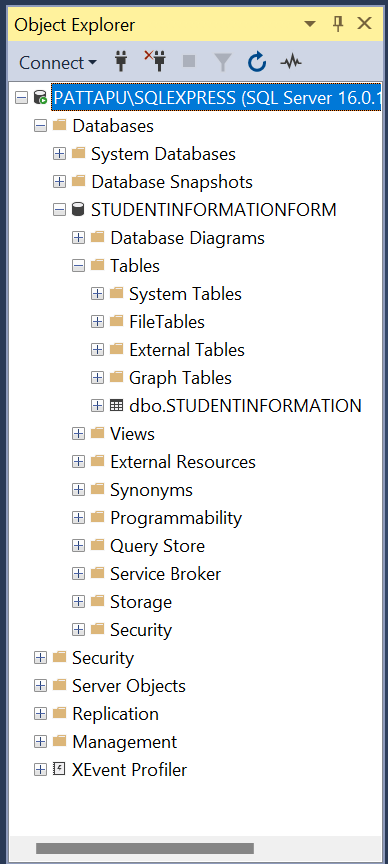
Step 5: Save the Table

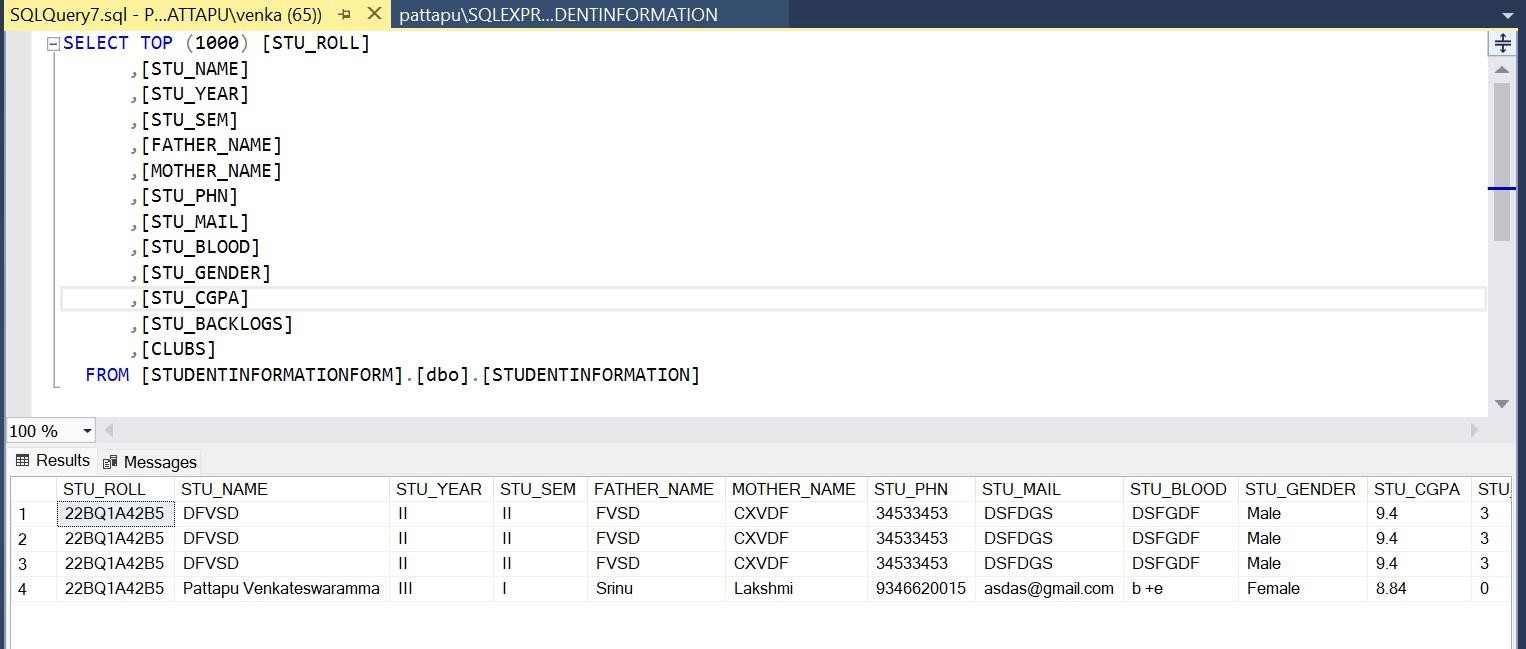
1. Once you’ve entered all the columns, right-click in the designer window and select Save Table (or press Ctrl + S).
2. You'll be prompted to name the table. Name it STUDENTINFORMATION and click OK. Step 6: Verify the Table
3. In the Object Explorer, right-click on the Tables folder under your database and select Refresh.
4. You should now see the STUDENTINFORMATION table listed.
5. Right-click on the table and select Top 1000 Rows to view the structure and verify the table creation.

Now your table is created and ready to store student information!



The above pictures are the creation of the table in the Microsoft sql server.





The above pictures indicates viewing the table data.

## Program:

using static System.Windows.Forms.VisualStyles.VisualStyleElement; using System.Data.SqlClient;

using static System.Runtime.InteropServices.JavaScript.JSType; using System.Drawing;

namespace Student\_Information\_lab\_2\_ { public partial class Form1 : Form {

string g;

public Form1() {

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e) { } if (stu\_gender\_male.Checked == true) {

g = "Male";

}

else if (stu\_gender\_female.Checked == true) { g = "Female";

} else{

g = "Other";

}

bool isAnyEmpty = false;

foreach (Control control in this.Controls) {

if (control is System.Windows.Forms.TextBox) { if (string.IsNullOrEmpty(control.Text) {

isAnyEmpty = true; break;

}

} else if (control is System.Windows.Forms.ComboBox) {

if (((System.Windows.Forms.ComboBox)control).SelectedIndex == -1) { isAnyEmpty = true;

break;

}

} else if (control is ListBox) {

if (((ListBox)control).SelectedIndex == -1) { isAnyEmpty = true;

break;

}

}

} if (!stu\_gender\_male.Checked && !stu\_gender\_female.Checked && !stu\_gender\_other.Checked) { isAnyEmpty = true;

} if (isAnyEmpty) {

MessageBox.Show("One or more fields are empty, fill it before submit");

} else {

string cs = "Data Source=PATTAPU\\SQLEXPRESS;Initial Catalog=STUDENTINFORMATIONFORM;Integrated Security=True";

SqlConnection con = new SqlConnection(cs); con.Open();

string insertQuery = "INSERT INTO STUDENTINFORMATION VALUES (@STU\_ROLL, @STU\_NAME, @STU\_YEAR, @STU\_SEM, @FATHER\_NAME, @MOTHER\_NAME, @STU\_PHN, @STU\_MAIL, @STU\_BLOOD, @STU\_GENDER, @STU\_CGPA, @STU\_BACKLOGS, @CLUBS)";

SqlCommand cmd = new SqlCommand(insertQuery, con);

cmd.Parameters.AddWithValue("@STU\_ROLL", sturoll.Text); cmd.Parameters.AddWithValue("@STU\_NAME", stuname.Text); cmd.Parameters.AddWithValue("@STU\_YEAR", stuyear.SelectedItem.ToString()); cmd.Parameters.AddWithValue("@STU\_SEM", stusem.SelectedItem.ToString()); cmd.Parameters.AddWithValue("@FATHER\_NAME", fathername.Text); cmd.Parameters.AddWithValue("@MOTHER\_NAME", mothername.Text); cmd.Parameters.AddWithValue("@STU\_PHN", stuphn.Text); cmd.Parameters.AddWithValue("@STU\_MAIL", stumail.Text); cmd.Parameters.AddWithValue("@STU\_BLOOD", stublood.Text); cmd.Parameters.AddWithValue("@STU\_GENDER", g); cmd.Parameters.AddWithValue("@STU\_CGPA", stucgpa.Text); cmd.Parameters.AddWithValue("@STU\_BACKLOGS", stubacklogs.Text); cmd.Parameters.AddWithValue("@CLUBS", clubs\_list.SelectedItem.ToString()); cmd.ExecuteNonQuery();

MessageBox.Show("Submission Complete", "info", MessageBoxButtons.OK, MessageBoxIcon.Information); ClearForm();

}

}

private void ClearForm() { sturoll.Text = string.Empty; stuname.Text = string.Empty; stuyear.SelectedIndex = -1;

stusem.SelectedIndex = -1; fathername.Text = string.Empty; mothername.Text = string.Empty; stuphn.Text = string.Empty; stumail.Text = string.Empty; stublood.Text = string.Empty; stucgpa.Text = string.Empty; stubacklogs.Text = string.Empty; clubs\_list.ClearSelected(); stu\_gender\_male.Checked = false; stu\_gender\_female.Checked = false; stu\_gender\_other.Checked = false;

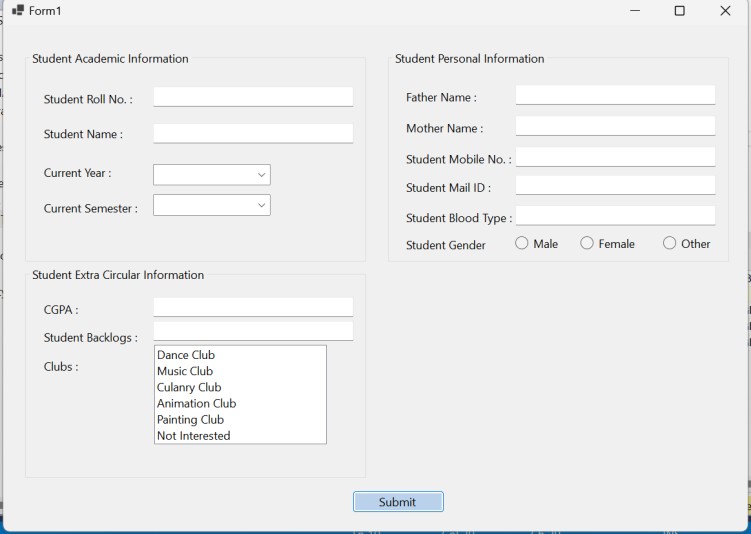
}

}

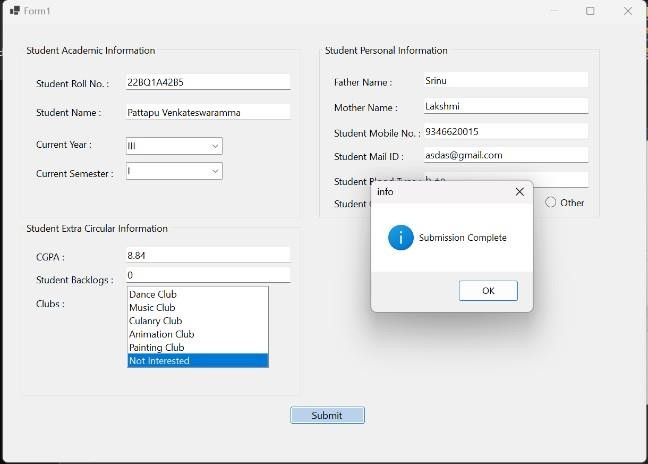
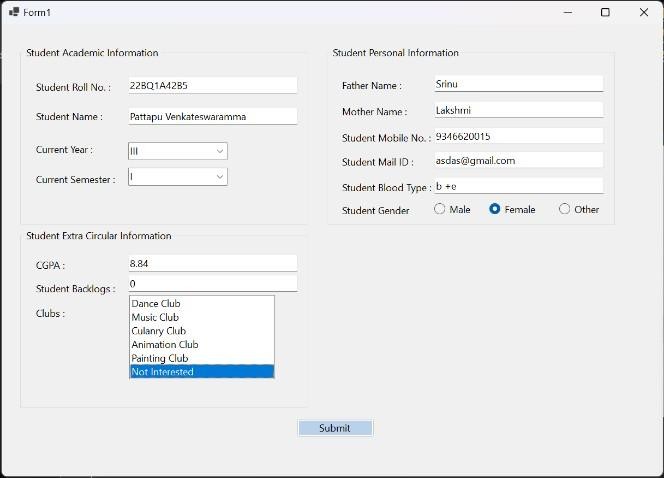
}

## Output:

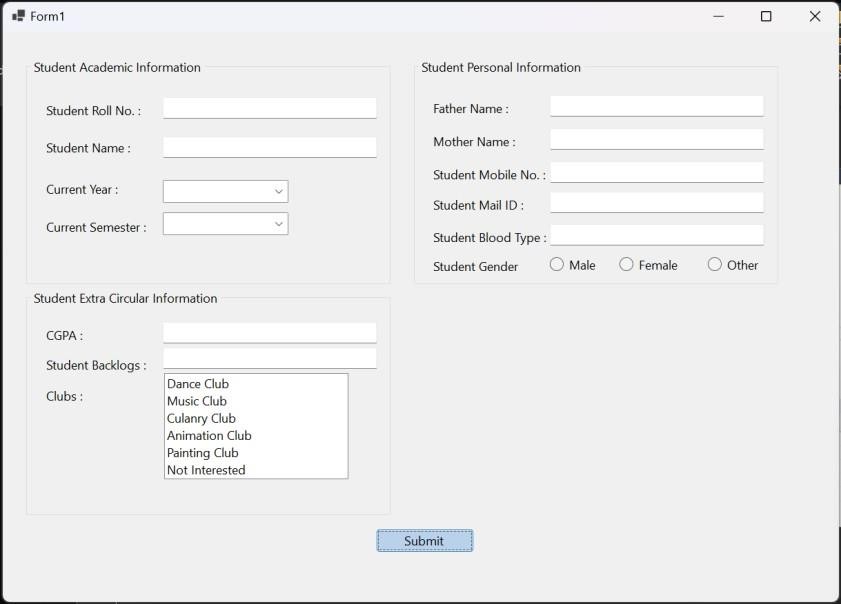
This is the created form of the Student Information.



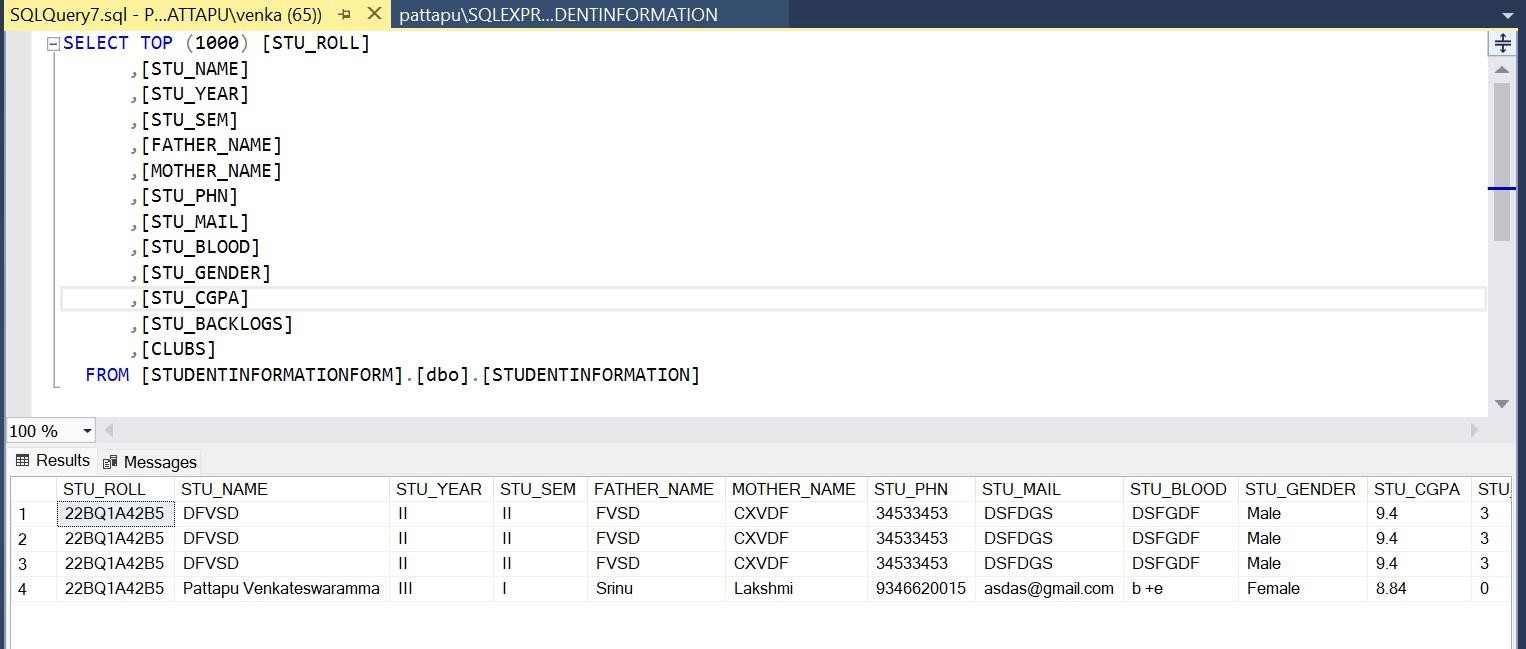
After filling the form with our details, click the submit button.



After clicking the ok button, the form will be refreshed to allow users to submit another response.

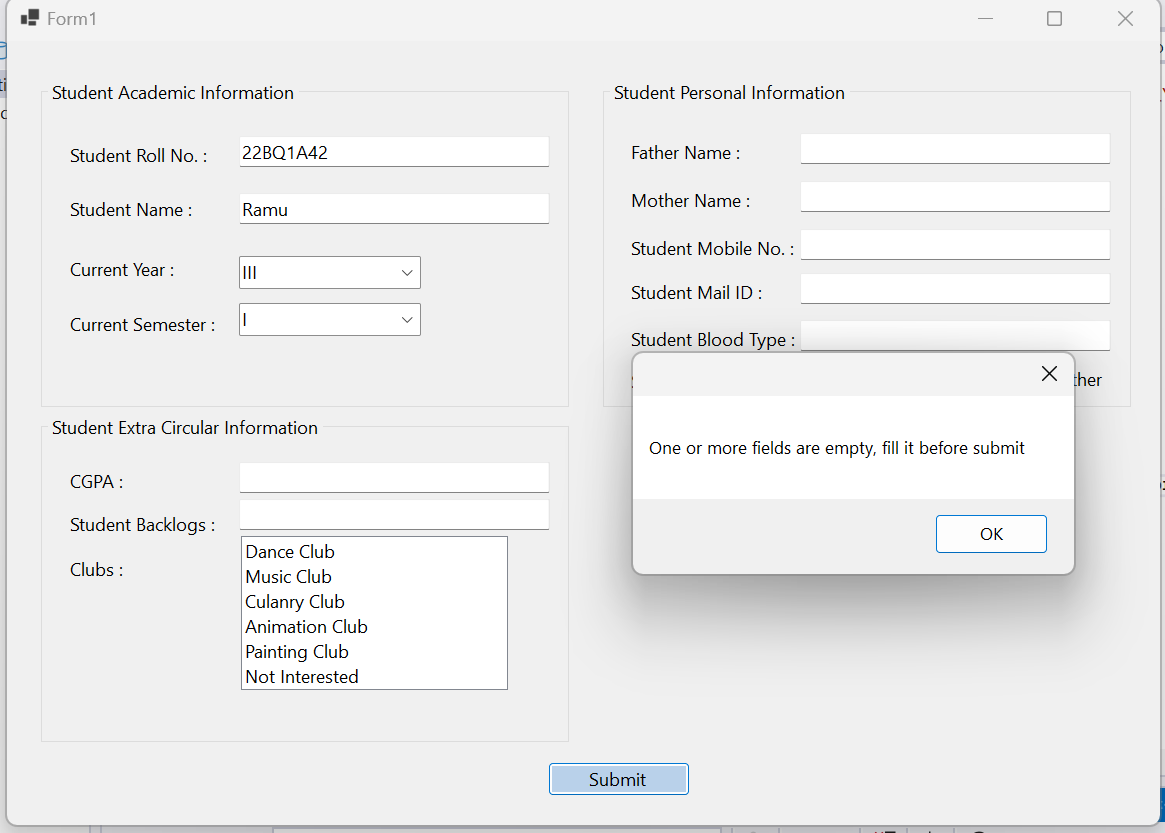


The database after submission of latest response for the Student Information form.



The latest submitted details will be at the end of the table.

If the user try to submit the form without providing all the details, then it will give an error message as follows



## Experiment Number:

**Date of Experiment:**

## Aim:

Overview of HTML, CSS, and JavaScript.

## HTML (HyperText Markup Language):

* **Purpose**: HTML is the foundational language for creating the structure of web pages. It defines the content and layout, marking different parts of a webpage such as headers, paragraphs, images, links, and more.

## Key Concepts:

* + **Tags**: HTML is written using elements enclosed in tags (e.g., <h1>, <p>, <a>). These tags define different content types, such as headings, paragraphs, links, and images.
  + **Attributes**: Tags can have attributes, which provide additional information. For example, the src attribute in an <img> tag defines the source of the image.
  + **Document Structure**: HTML documents start with a <!DOCTYPE html> declaration, followed by a <html> tag that contains two main sections: the <head> (metadata like title, CSS links) and the <body> (visible content on the webpage).
  + **Semantic HTML**: Newer HTML5 introduced semantic tags like <header>, <footer>, <article>, and

<section>, which help improve accessibility and SEO by clearly defining the role of each content block.

* **Examples**: html

Copy code

<!DOCTYPE html>

<html>

<head>

<title>My Web Page</title>

</head>

<body>

<h1>Welcome to My Website</h1>

<p>This is a simple paragraph on the page.</p>

<a href="https://example.com">Visit Example</a>

</body>

</html>

## CSS (Cascading Style Sheets):

* **Purpose**: CSS is used to style the visual presentation of a webpage created with HTML. It controls the layout, color schemes, fonts, and responsiveness of the webpage across different devices.

## Key Concepts:

* + **Selectors**: These are used to target specific HTML elements for styling. For example, you can style an

<h1> or a class .button using selectors.

* + **Properties and Values**: CSS consists of properties (e.g., color, font-size, margin) and corresponding values that define the look of an element. For example, color: blue; changes the text color to blue.
  + **Box Model**: Every HTML element can be thought of as a box. The box model consists of margins, borders, padding, and the content itself, allowing you to control the spacing around elements.
  + **Responsive Design**: CSS is essential for creating responsive websites that adjust layout according to screen size. Media queries (@media) are used to apply different styles based on device characteristics (e.g., width, orientation).
  + **CSS Frameworks**: Tools like Bootstrap and Tailwind CSS simplify styling by offering pre-built classes for common layouts and design patterns.
* **Examples**: css

Copy code h1 {

color: blue; font-size: 24px;

text-align: center;

}

p {

margin: 10px;

line-height: 1.5;

}

@media (max-width: 600px) { h1 {

font-size: 18px;

}

}

## JavaScript (JS):

* **Purpose**: JavaScript is a dynamic programming language that enables interactive and functional elements on websites. It is responsible for client-side logic, event handling, and dynamic content updates without reloading the webpage.

## Key Concepts:

* + **Variables and Data Types**: JavaScript uses variables to store data, which can be of different types such as strings, numbers, objects, and arrays.
  + **Functions**: Functions are reusable blocks of code that perform specific tasks. JavaScript functions can take inputs (parameters) and return outputs.
  + **DOM Manipulation**: The **Document Object Model (DOM)** is a representation of the HTML structure. JavaScript can interact with and manipulate the DOM to dynamically change content, add or remove elements, and modify styles in response to user actions.
  + **Events**: JavaScript allows websites to respond to user interactions (clicks, keyboard input, mouse

movements) through events. For example, a button click may trigger a function to display a message or submit a form.

* + **Asynchronous Programming**: JavaScript supports asynchronous operations (e.g., making API requests, loading content in the background) using techniques like **callbacks**, **promises**, and **async/await**. This allows for smoother user experiences, where tasks are completed without blocking other operations.
  + **Frameworks and Libraries**: Popular frameworks like **React**, **Angular**, and **Vue.js** simplify JavaScript development by providing structure and reusable components. Libraries like **jQuery** help with DOM manipulation and event handling.
* **Examples**: javascript Copy code

// Variables and Functions const name = "John"; function greet() {

alert("Hello, " + name + "!");

}

// Event Listener

document.querySelector("button").addEventListener("click", greet);

// DOM Manipulation

document.querySelector("h1").style.color = "blue";

## Overview of Forms in HTML

**HTML Forms** are essential for collecting user input and submitting it to the server for processing. They consist of various input fields where users can enter data, such as text, passwords, emails, or select options.

* **Form Tag**: The <form> tag wraps the input elements and defines how the data is submitted, specifying where to send the data and what method to use (GET or POST).

## Common Form Elements:

* + **Text Input**: Single-line input field used for text entries, such as names or search queries.
  + **Password Input**: Similar to text input, but it hides the characters entered.
  + **Radio Buttons**: Allow users to select one option from multiple choices.
  + **Checkboxes**: Allow users to select multiple options from a list.
  + **Submit Button**: A button that, when clicked, sends the form data to the server.
  + **Dropdown (Select)**: A drop-down menu that offers a list of options for users to choose from.
  + **Textarea**: A multi-line text field for longer user input, like comments or feedback.

Forms are crucial for tasks such as user authentication, collecting feedback, and managing data input on websites.

## Overview of Bootstrap

**Bootstrap** is a widely-used front-end framework that simplifies the development of responsive, mobile-first websites. It includes pre-designed components and styles for building modern web pages quickly.

* **Responsive Grid System**: Bootstrap uses a flexible 12-column grid layout that adapts to different screen sizes, making it easier to create responsive designs.

## Pre-built Components:

* + **Buttons**: Stylish and customizable buttons with various types, sizes, and colors.
  + **Forms**: Provides well-designed form elements such as input fields, checkboxes, and dropdowns, making forms look cleaner and more user-friendly.
  + **Navbars**: Ready-made responsive navigation bars for building site menus, which adjust automatically on different devices.
* **Customizable and Extensible**: Bootstrap is highly customizable, allowing developers to modify styles and themes to fit their projects.
* **Cross-browser Compatibility**: Ensures consistent appearance across various browsers and devices.

Bootstrap greatly enhances the speed and ease of web development by offering a vast collection of ready-to-use components, reducing the need to build everything from scratch.

documentation for Bootstrap can be found at: [https://getbootstrap.com](https://getbootstrap.com/)

This site provides detailed guides, examples, and API references for all Bootstrap components, utilities, and customization options.

## Program:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Student Form</title>

<link rel="stylesheet" href="style.css">

<link href=["https](https://cdn.jsdelivr.net/npm/bootstrap%405.3.3/dist/css/bootstrap.min.css):[//cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"](https://cdn.jsdelivr.net/npm/bootstrap%405.3.3/dist/css/bootstrap.min.css) rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">

</head>

<body>

<header>

<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">

<div class="container">

<img src="images/vvit\_logo.png" alt="vvit-logo" height="100px" width="140px">

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"

aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">

<ul class="navbar-nav flex-grow-1">

<li class="nav-item">

<a class="nav-link text-dark" href="#" onclick="showContent('welcome')">Home</a>

</li>

<li class="nav-item">

<a class="nav-link text-dark" href="#" onclick="showContent('studentForm')">Student Form</a>

</li>

<li class="nav-item">

<a class="nav-link text-dark" href="#" onclick="showContent('facultyForm')">Faculty Form</a>

</li>

<li class="nav-item">

<a class="nav-link text-dark" href="#" onclick="showContent('sacForm')">Sca Form</a>

</li>

<li class="nav-item">

<a class="nav-link text-dark" href="#" onclick="showContent('libraryForm')">Library Form</a>

</li>

<li class="nav-item"></li>

<a class="nav-link text-dark" href="#">Log in</a>

</li>

</ul>

</div>

</div>

</nav>

</header>

<div class="container-fluid">

<div class="row">

<!-- Left column (30%) -->

<div class="col-md-3 bg-light p-3" id="title-column" style="height: 80vh; overflow-y: auto;">

<h3>Navigation</h3>

<ul class="list-group">

<li class="list-group-item">

<a href="#" onclick="showContent('welcome')"> Welcome</a>

</li>

<li class="list-group-item">

<a href="#" onclick="showContent('studentForm')">Student Form</a>

</li>

<li class="list-group-item">

<a href="#" onclick="showContent('facultyForm')">Faculty Form</a>

</li>

<li class="list-group-item">

<a href="#" onclick="showContent('sacForm')">Sca Form</a>

</li>

<li class="list-group-item">

<a href="#" onclick="showContent('libraryForm')">Library Form</a>

</li>

<li class="list-group-item">

<a href="#" onclick="showContent('aboutSection')">About Me</a>

</li>

</ul>

</div>

<!-- Right column (70%) -->

<div class="col-md-9 p-3" id="content-column" style="height: 80vh; overflow-y: auto;">

<div id="welcome" class="content-section">

<h3>WELCOME</h3>

<p>Need to register as a student? Click here: <a href="#" onclick="showContent('studentForm')">Student Form</a></p>

</div>

<!-- Student Form Section -->

<div id="studentProfileForm" class="profile-form-section" style="display:none;">

<h3>Student Profile</h3>

<form id="studentForm" onsubmit="switchToAbout(event)">

<div class="form-group">

<label for="studentName">Student Name:</label>

<input type="text" id="studentName" name="studentName" class="form-control" required />

</div>

<div class="form-group">

<label for="registerNo">Register No:</label>

<input type="text" id="registerNo" name="registerNo" class="form-control" required />

</div>

<div class="form-group">

<label for="branch">Branch:</label>

<select id="branch" name="branch" class="form-control">

<option value="--select--">--select--</option>

<option value="CSE">CSE</option>

<option value="CSM">CSM</option>

<option value="CSO">CSO</option>

<option value="CIC">CIC</option>

<option value="IT">IT</option>

<option value="AI&ML">AI&ML</option>

<option value="AI&DS">AI&DS</option>

<option value="ECE">ECE</option>

<option value="EEE">EEE</option>

</select>

</div>

<div class="form-group">

<label for="semester">Semester:</label>

<input type="number" id="semester" name="semester" class="form-control" required />

</div>

<div class="form-group">

<label for="email">Email-id:</label>

<input type="email" id="email" name="email" class="form-control" required />

</div>

<div class="form-group">

<label for="phone">Phone No:</label>

<input type="text" id="phone" name="phone" class="form-control" required />

</div>

<div class="form-group">

<label for="fatherName">Father Name:</label>

<input type="text" id="fatherName" name="fatherName" class="form-control" required />

</div>

<div class="form-group">

<label for="fatherPhone">Father Phone No:</label>

<input type="text" id="fatherPhone" name="fatherPhone" class="form-control" required />

</div>

<div class="form-group">

<label for="motherName">Mother Name:</label>

<input type="text" id="motherName" name="motherName" class="form-control" required />

</div>

<div class="form-group">

<label for="motherPhone">Mother Phone No:</label>

<input type="text" id="motherPhone" name="motherPhone" class="form-control" required />

</div>

<div class="form-group">

<label for="bloodGroup">Blood Group:</label>

<input type="text" id="bloodGroup" name="bloodGroup" class="form-control" />

</div>

<div class="form-group">

<label>Gender:</label><br />

<input type="radio" id="male" name="gender" value="Male" /> Male

<input type="radio" id="female" name="gender" value="Female" /> Female

</div>

<div class="form-group">

<label for="address">Address:</label>

<textarea id="address" name="address" class="form-control"></textarea>

</div>

<div class="form-group">

<label for="skills">Skills:</label>

<textarea id="skills" name="skills" class="form-control"></textarea>

</div>

<button type="submit" class="btn-submit">Submit</button>

</form>

</div>

<!-- Faculty Form Section -->

<div id="facultyProfileForm" class="profile-form-section" style="display:none;">

<h3>Faculty Profile</h3>

<form id="facultyForm">

<div class="form-group">

<label for="facultyName">Faculty Name:</label>

<input type="text" id="facultyName" name="facultyName" class="form-control" required />

</div>

<div class="form-group">

<label for="facultyRegisterNo">Register No:</label>

<input type="text" id="facultyRegisterNo" name="facultyRegisterNo" class="form-control" required />

</div>

<div class="form-group">

<label for="department">Department:</label>

<select id="department" name="department" class="form-control">

<option value="--select--">--select--</option>

<option value="CSE">CSE</option>

<option value="CSM">CSM</option>

<option value="CSO">CSO</option>

<option value="CIC">CIC</option>

<option value="IT">IT</option>

<option value="AI&ML">AI&ML</option>

<option value="AI&DS">AI&DS</option>

<option value="ECE">ECE</option>

<option value="EEE">EEE</option>

</select>

</div>

<div class="form-group">

<label for="facultyEmail">Email-id:</label>

<input type="email" id="facultyEmail" name="facultyEmail" class="form-control" required />

</div>

<div class="form-group">

<label for="facultyPhone">Phone No:</label>

<input type="text" id="facultyPhone" name="facultyPhone" class="form-control" required />

</div>

<div class="form-group">

<label for="facultyBloodGroup">Blood Group:</label>

<input type="text" id="facultyBloodGroup" name="facultyBloodGroup" class="form-control" />

</div>

<div class="form-group">

<label>Gender:</label><br />

<input type="radio" id="facultyMale" name="gender" value="Male" /> Male

<input type="radio" id="facultyFemale" name="gender" value="Female" /> Female

</div>

<div class="form-group">

<label for="facultyAddress">Address:</label>

<textarea id="facultyAddress" name="facultyAddress" class="form-control"></textarea>

</div>

<div class="form-group">

<label for="subjectTeaches">Subject Teaches:</label>

<textarea id="subjectTeaches" name="subjectTeaches" class="form-control"></textarea>

</div>

<button type="submit" class="btn-submit">Submit</button>

</form>

</div>

<!-- SAC Form Section -->

<div id="sacProfileForm" class="profile-form-section" style="display:none;">

<h3>SAC Membership</h3>

<form id="sacForm">

<div class="form-group">

<label>Member in SAC:</label><br />

<input type="radio" id="sacYes" name="sacMember" value="Yes" /> Yes

<input type="radio" id="sacNo" name="sacMember" value="No" /> No

</div>

<div class="form-group">

<label for="councilName">Council Name:</label>

<textarea id="councilName" name="councilName" class="form-control"></textarea>

</div>

<button type="submit" class="btn-submit">Submit</button>

</form>

</div>

<!-- Library Form Section -->

<div id="libraryProfileForm" class="profile-form-section" style="display:none;">

<h3>Library Details</h3>

<form id="libraryForm">

<div class="form-group">

<label for="libraryID">Library ID:</label>

<input type="text" id="libraryID" name="libraryID" class="form-control" required />

</div>

<div class="form-group">

<label for="booksBorrowed">Books Borrowed:</label>

<textarea id="booksBorrowed" name="booksBorrowed" class="form-control"></textarea>

</div>

<button type="submit" class="btn-submit">Submit</button>

</form>

</div>

<!-- About Section -->

<div id="aboutSection" class="content-section" style="display:none;">

<h3>About the Student</h3>

<p><strong>Student Name:</strong> <span id="displayStudentName"></span></p>

<p><strong>Register No:</strong> <span id="displayRegisterNo"></span></p>

<p><strong>Branch:</strong> <span id="displayBranch"></span></p>

<p><strong>Semester:</strong> <span id="displaySemester"></span></p>

<p><strong>Email:</strong> <span id="displayEmail"></span></p>

<p><strong>Phone No:</strong> <span id="displayPhone"></span></p>

<p><strong>Father Name:</strong> <span id="displayFatherName"></span></p>

<p><strong>Father Phone No:</strong> <span id="displayFatherPhone"></span></p>

<p><strong>Mother Name:</strong> <span id="displayMotherName"></span></p>

<p><strong>Mother Phone No:</strong> <span id="displayMotherPhone"></span></p>

<p><strong>Blood Group:</strong> <span id="displayBloodGroup"></span></p>

<p><strong>Gender:</strong> <span id="displayGender"></span></p>

<p><strong>Address:</strong> <span id="displayAddress"></span></p>

<p><strong>Skills:</strong> <span id="displaySkills"></span></p>

</div>

</div>

</div>

</div>

<script>

function showContent(sectionId) {

var sections = document.getElementsByClassName('content-section'); for (var i = 0; i < sections.length; i++) {

sections[i].style.display = 'none';

}

document.getElementById(sectionId).style.display = 'block';

}

function switchToAbout(event) { event.preventDefault();

// Fetching form values

document.getElementById('displayStudentName').textContent = document.getElementById('StudentName').value;

document.getElementById('displayRegisterNo').textContent = document.getElementById('RegisterNo').value; document.getElementById('displayBranch').textContent = document.getElementById('Branch').value; document.getElementById('displaySemester').textContent = document.getElementById('Semester').value; document.getElementById('displayEmail').textContent = document.getElementById('Email').value; document.getElementById('displayPhone').textContent = document.getElementById('Phone').value;

document.getElementById('displayFatherName').textContent = document.getElementById('FatherName').value;

document.getElementById('displayFatherPhone').textContent = document.getElementById('FatherPhone').value;

document.getElementById('displayMotherName').textContent = document.getElementById('MotherName').value;

document.getElementById('displayMotherPhone').textContent = document.getElementById('MotherPhone').value;

document.getElementById('displayBloodGroup').textContent = document.getElementById('BloodGroup').value; var gender = document.querySelector('input[name="Gender"]:checked');

if (gender) {

document.getElementById('displayGender').textContent = gender.value;

}

document.getElementById('displayAddress').textContent = document.getElementById('Address').value; document.getElementById('displaySkills').textContent = document.getElementById('Skills').value; showContent('aboutSection');

}

</script> </body> </html>

## Output:

