Calculator Application in C#

Student Name: W. A. Kawshan Fernando

Reg. No: 0303861

Date of Submission: 10/11/2025

Introduction

This project is about creating a simple Calculator Application using C# Windows Form. The calculator is designed to perform the four basic arithmetic operations: addition, subtraction, multiplication, and division. It allows users to enter two numbers and select an operation to get the result instantly. The program provides a clean and easy-to-use graphical interface, making it suitable for anyone who needs quick and accurate calculations.

The main purpose of this project is to understand how to design and develop a basic desktop application using Windows Forms in C#. It helps to practice event-driven programming, user interface design, and logical problem-solving. The scope of the calculator is limited to handling simple arithmetic expressions involving two numbers, which makes it ideal for learning the fundamentals of GUI development and control handling in C#.

The programming language used in this project is **C#**. It was chosen because it is powerful, easy to learn, and well-suited for building Windows applications. C# provides built-in libraries for graphical interfaces, event handling, and data validation, which makes the development process efficient. In addition, Visual Studio offers a user-friendly environment for designing forms and writing, testing, and debugging the code effectively.

Problem Analysis

The main problem addressed by this project is the **need for a simple and user-friendly calculator** that can quickly perform basic arithmetic operations such as addition, subtraction, multiplication, and division. Many users prefer a lightweight tool that does not require complex steps or internet access, especially for quick day-to-day calculations. Therefore, this project focuses on building a small desktop calculator that anyone can easily use.

User Requirements

To solve the problem effectively, the calculator must meet the following user requirements:

- Allow the user to **input two numbers**.
- Provide buttons for **basic arithmetic operations** (+, −, ×, ÷).
- Display the **result clearly** on the screen.
- Include options to **clear** the input and start a new calculation.
- Handle **invalid inputs** or **division by zero** gracefully by showing an error message.

Functional Requirements

These define what the calculator must do:

1. Accept numeric inputs from the user.
2. Perform addition, subtraction, multiplication, and division.
3. Display the calculated result after pressing the "=" button.
4. Reset all fields when the "Clear" button is clicked.
5. Validate user inputs and show error messages when necessary (e.g., dividing by zero).

Non-Functional Requirements

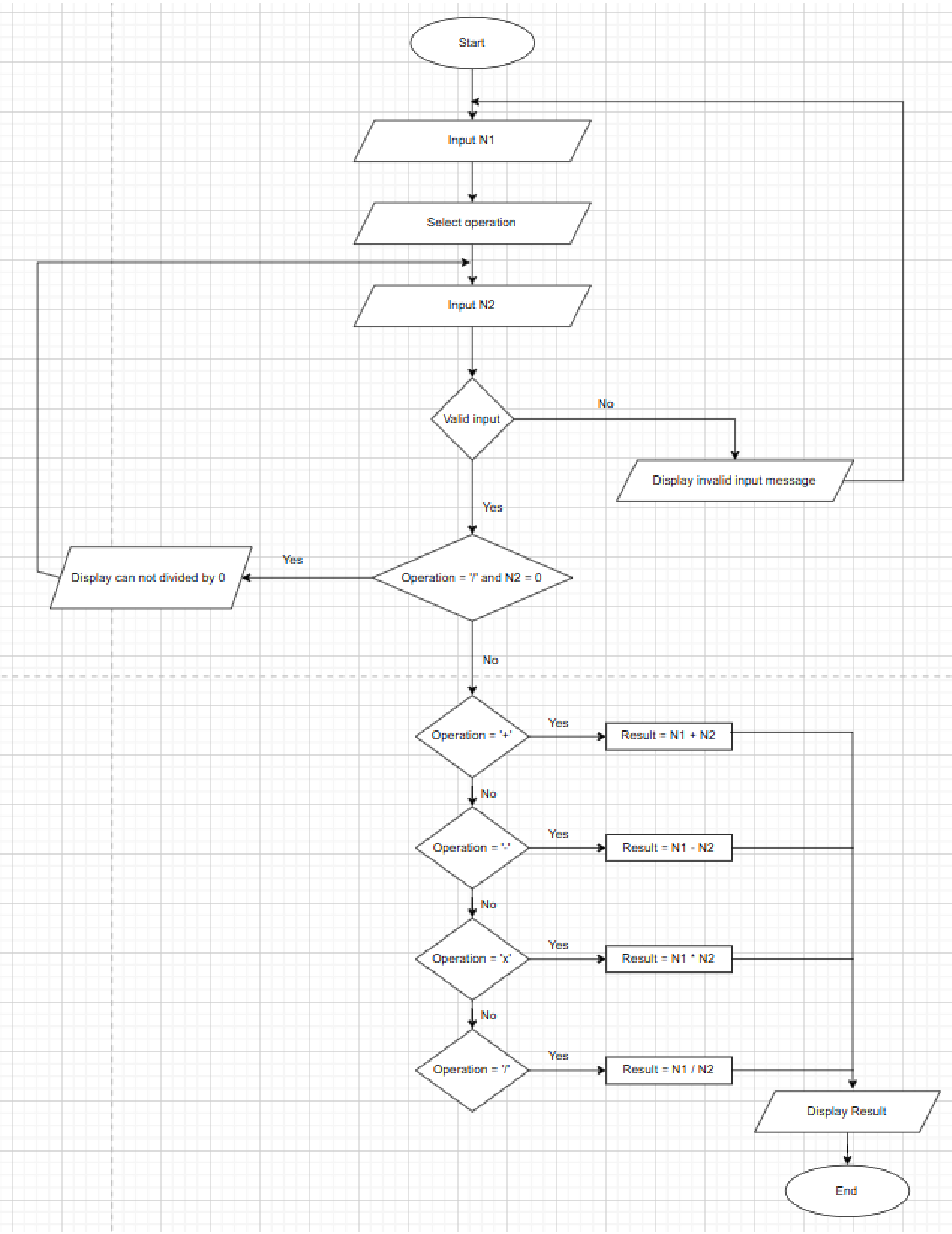
These describe how the system should perform:

1. **Usability:** The interface should be simple, clean, and easy to use even for beginners.
2. **Reliability:** The calculator should provide accurate results without crashing or freezing.
3. **Performance:** Calculations and UI responses should be processed instantly.
4. **Maintainability:** The code should be organized and easy to update or extend in the future.
5. **Portability:** The application should run smoothly on any Windows computer with .NET support.

In summary, the calculator solves a common need for quick and reliable arithmetic computation while helping to understand key programming concepts such as event handling, user input validation, and GUI design in C#.

System Design

A) Flowchart

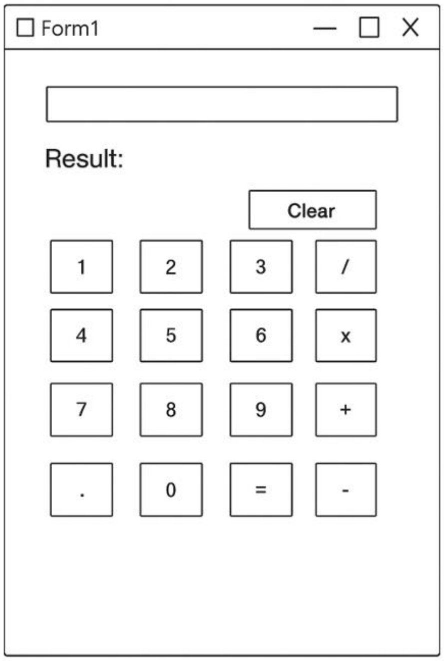


B) Pseudocode

```
Begin

INPUT N1
SELECT operation
INPUT N2

IF (N1 or N2 is invalid) THEN
   DISPLAY "Invalid input message"
   GO BACK to input
ELSE
   IF (op == '/' AND N2 == 0) THEN
      DISPLAY "Cannot divide by zero"
      GO BACK to input
   ELSE
      IF (op == '+') THEN
         Result = N1 + N2
      ELSE IF (op == '-') THEN
         Result = N1 - N2
      ELSE IF (op == '*') THEN
```

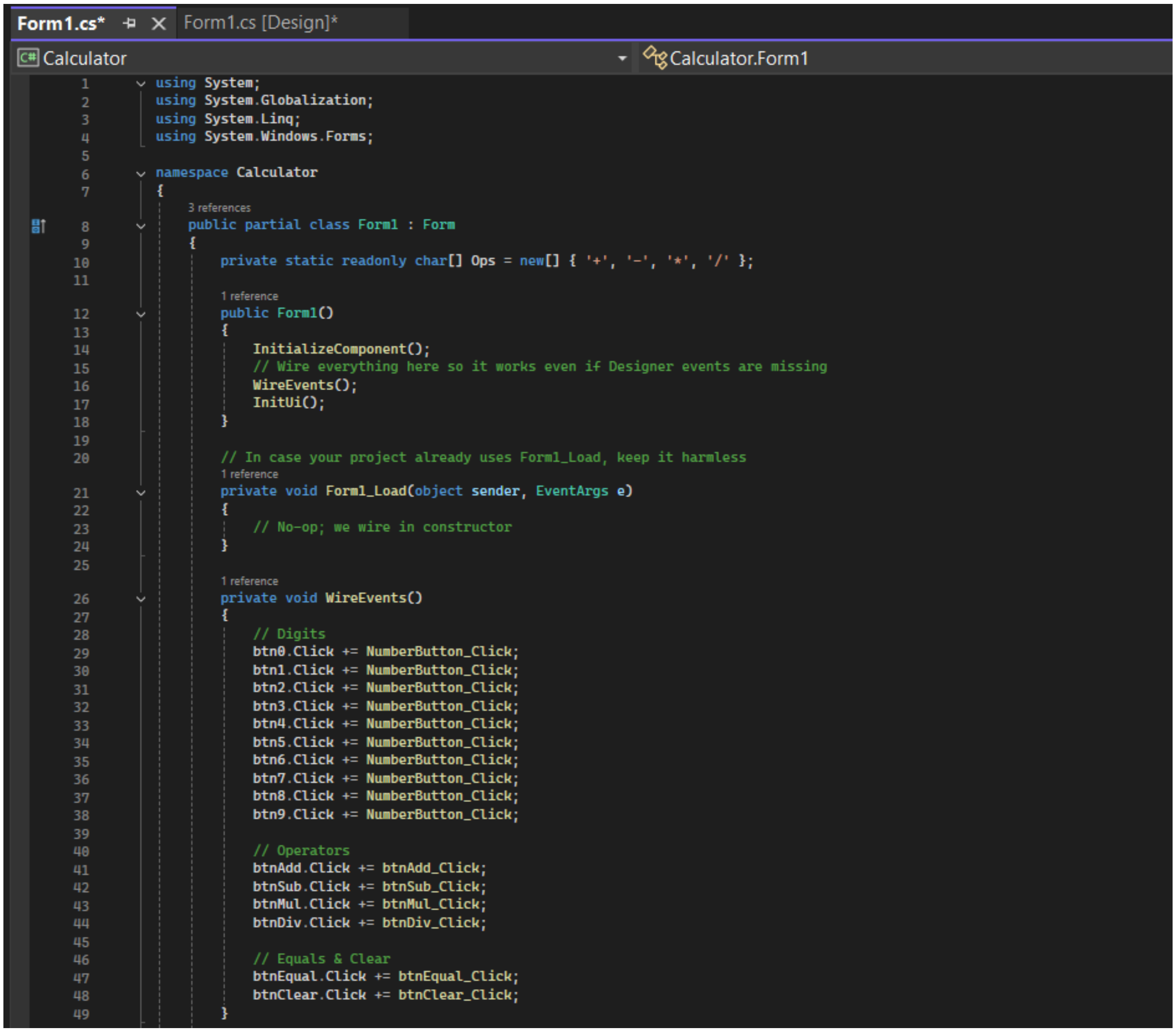
```
            Result = N1 * N2
        ELSE IF (op == '/') THEN
            Result = N1 / N2
        ENDIF
        DISPLAY Result
    ENDIF
ENDIF

END
```

C) Wireframe

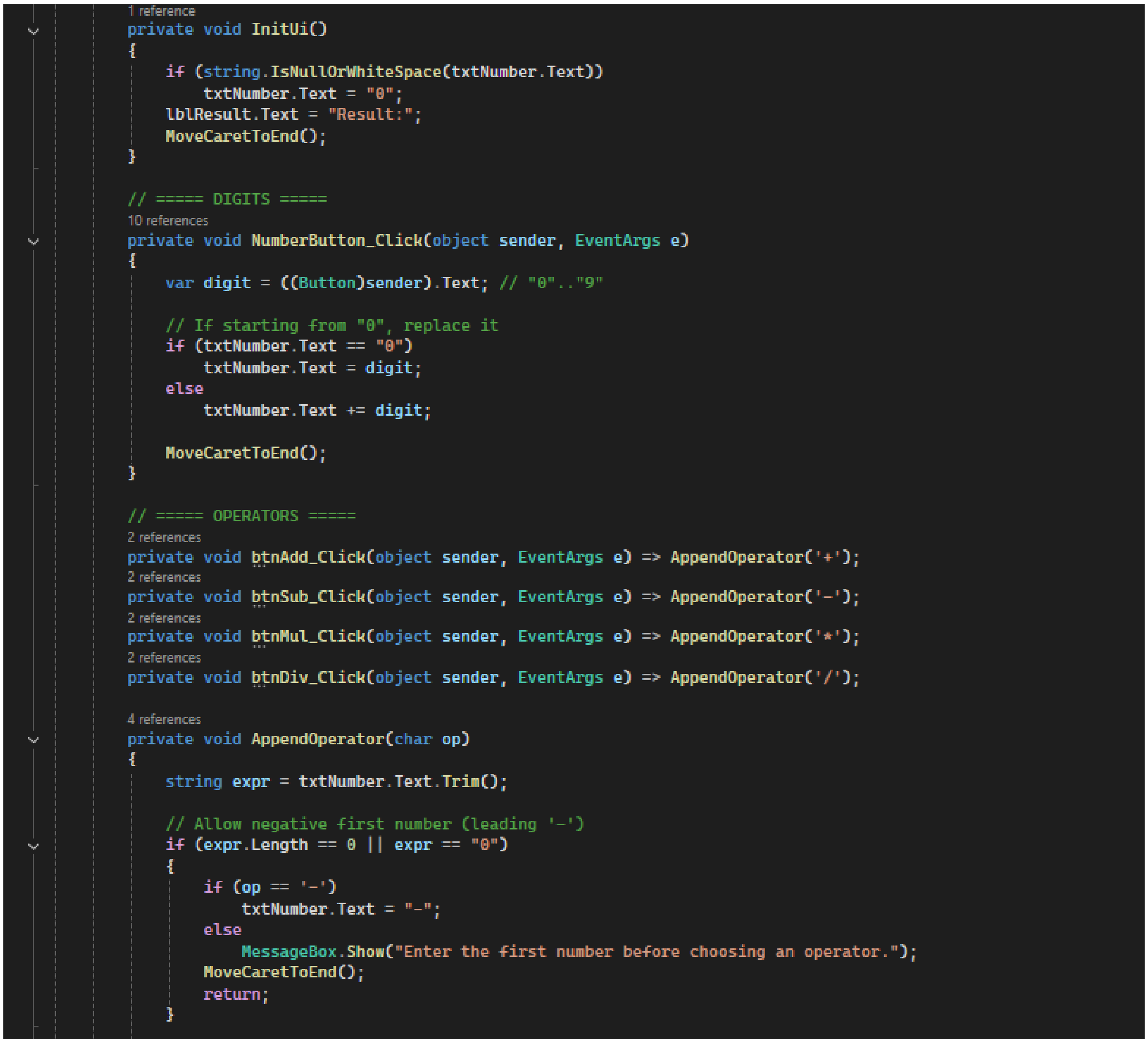


Implementation

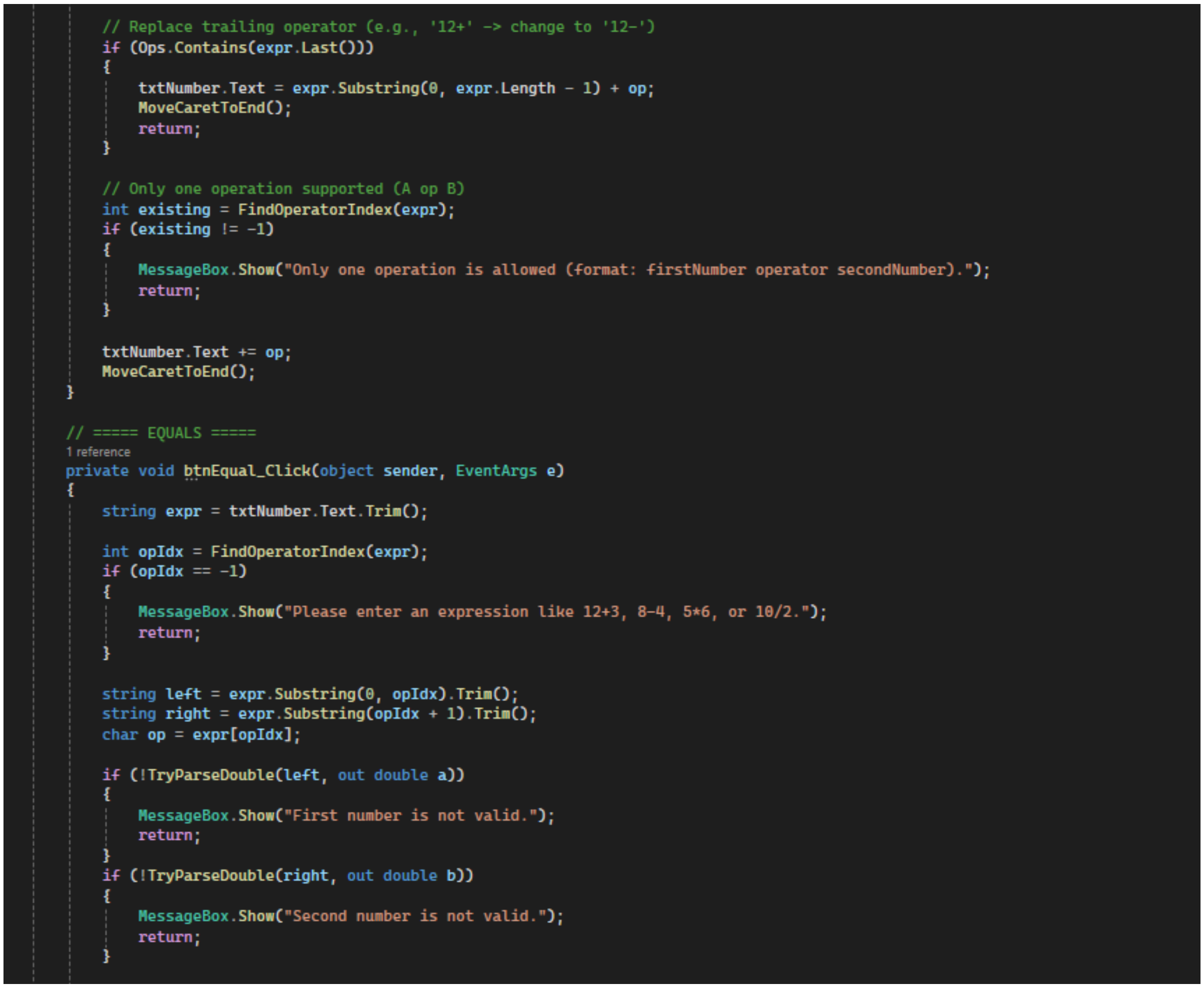
C# code snippets



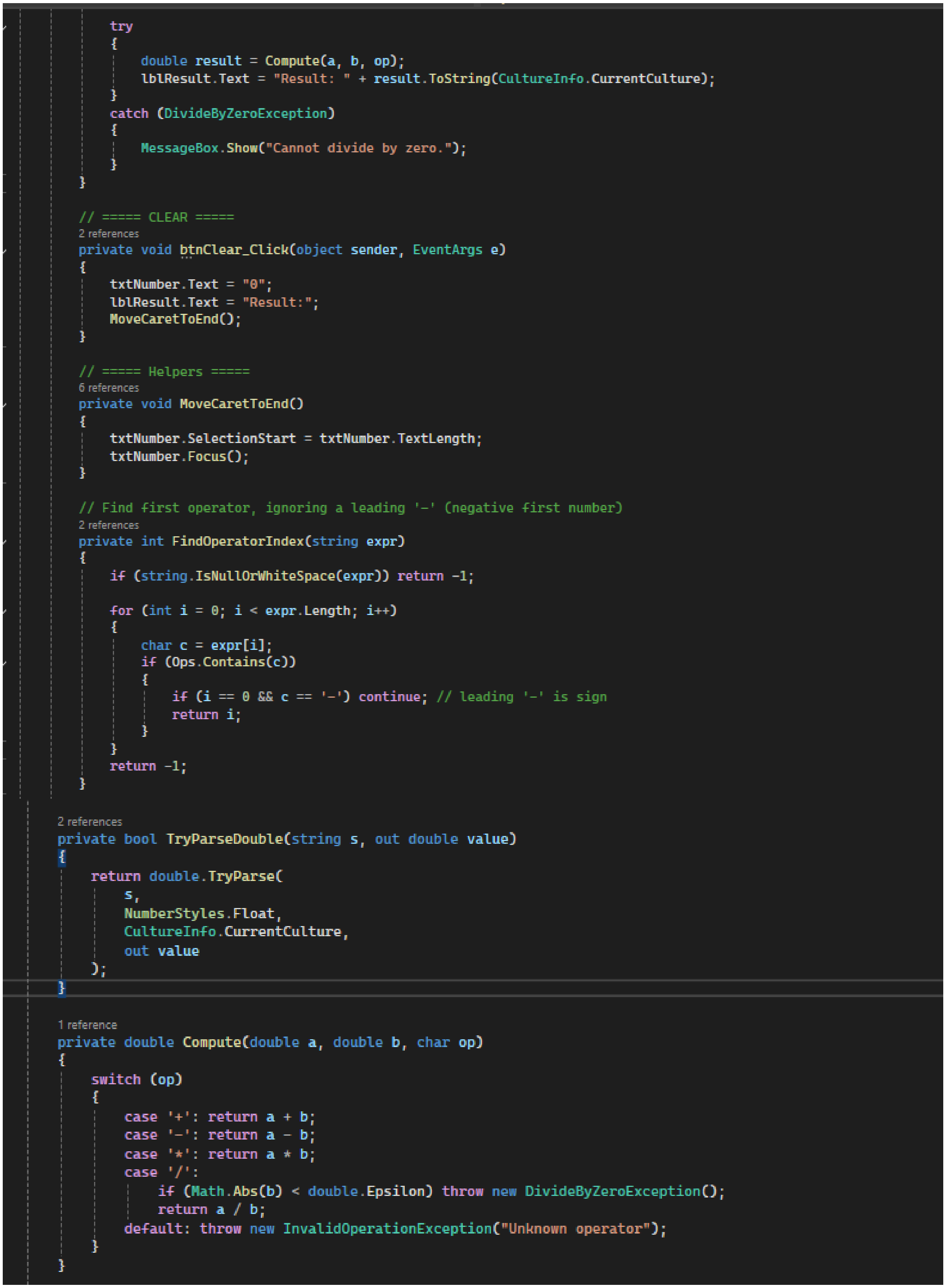
```
using System;
using System.Globalization;
using System.Linq;
using System.Windows.Forms;

namespace Calculator
{
    3 references
    public partial class Form1 : Form
    {
        private static readonly char[] Ops = new[] { '+', '-', '*', '/' };

        1 reference
        public Form1()
        {
            InitializeComponent();
            // Wire everything here so it works even if Designer events are missing
            WireEvents();
            InitUi();
        }

        // In case your project already uses Form1_Load, keep it harmless
        1 reference
        private void Form1_Load(object sender, EventArgs e)
        {
            // No-op; we wire in constructor
        }

        1 reference
        private void WireEvents()
        {
            // Digits
            btn0.Click += NumberButton_Click;
            btn1.Click += NumberButton_Click;
            btn2.Click += NumberButton_Click;
            btn3.Click += NumberButton_Click;
            btn4.Click += NumberButton_Click;
            btn5.Click += NumberButton_Click;
            btn6.Click += NumberButton_Click;
            btn7.Click += NumberButton_Click;
            btn8.Click += NumberButton_Click;
            btn9.Click += NumberButton_Click;

            // Operators
            btnAdd.Click += btnAdd_Click;
            btnSub.Click += btnSub_Click;
            btnMul.Click += btnMul_Click;
            btnDiv.Click += btnDiv_Click;

            // Equals & Clear
            btnEqual.Click += btnEqual_Click;
            btnClear.Click += btnClear_Click;
        }
```



```
1 reference
private void InitUi()
{
    if (string.IsNullOrWhiteSpace(txtNumber.Text))
        txtNumber.Text = "0";
    lblResult.Text = "Result:";
    MoveCaretToEnd();
}

// ===== DIGITS =====
10 references
private void NumberButton_Click(object sender, EventArgs e)
{
    var digit = ((Button)sender).Text; // "0".."9"

    // If starting from "0", replace it
    if (txtNumber.Text == "0")
        txtNumber.Text = digit;
    else
        txtNumber.Text += digit;

    MoveCaretToEnd();
}

// ===== OPERATORS =====
2 references
private void btnAdd_Click(object sender, EventArgs e) => AppendOperator('+');
2 references
private void btnSub_Click(object sender, EventArgs e) => AppendOperator('-');
2 references
private void btnMul_Click(object sender, EventArgs e) => AppendOperator('*');
2 references
private void btnDiv_Click(object sender, EventArgs e) => AppendOperator('/');

4 references
private void AppendOperator(char op)
{
    string expr = txtNumber.Text.Trim();

    // Allow negative first number (leading '-')
    if (expr.Length == 0 || expr == "0")
    {
        if (op == '-')
            txtNumber.Text = "-";
        else
            MessageBox.Show("Enter the first number before choosing an operator.");
        MoveCaretToEnd();
        return;
    }
```

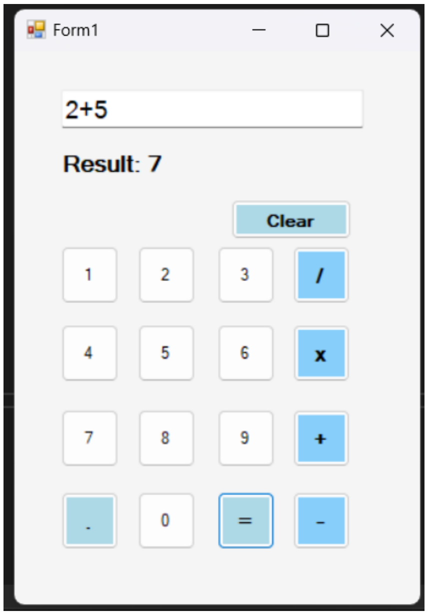


```
            // Replace trailing operator (e.g., '12+' -> change to '12-')
            if (Ops.Contains(expr.Last()))
            {
                txtNumber.Text = expr.Substring(0, expr.Length - 1) + op;
                MoveCaretToEnd();
                return;
            }

            // Only one operation supported (A op B)
            int existing = FindOperatorIndex(expr);
            if (existing != -1)
            {
                MessageBox.Show("Only one operation is allowed (format: firstNumber operator secondNumber).");
                return;
            }

            txtNumber.Text += op;
            MoveCaretToEnd();
        }

        // ===== EQUALS =====
        1 reference
        private void btnEqual_Click(object sender, EventArgs e)
        {
            string expr = txtNumber.Text.Trim();

            int opIdx = FindOperatorIndex(expr);
            if (opIdx == -1)
            {
                MessageBox.Show("Please enter an expression like 12+3, 8-4, 5*6, or 10/2.");
                return;
            }

            string left = expr.Substring(0, opIdx).Trim();
            string right = expr.Substring(opIdx + 1).Trim();
            char op = expr[opIdx];

            if (!TryParseDouble(left, out double a))
            {
                MessageBox.Show("First number is not valid.");
                return;
            }
            if (!TryParseDouble(right, out double b))
            {
                MessageBox.Show("Second number is not valid.");
                return;
            }
```



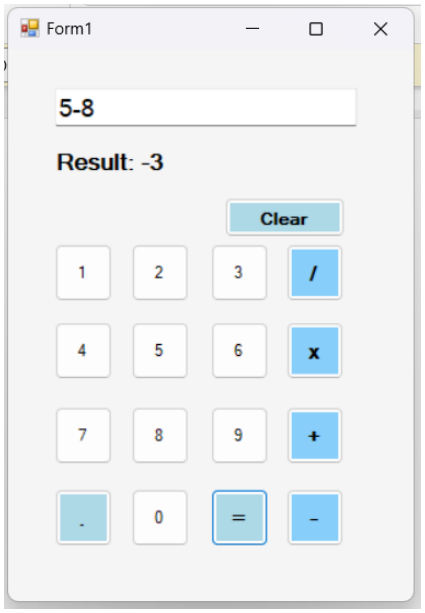
```
            try
            {
                double result = Compute(a, b, op);
                lblResult.Text = "Result: " + result.ToString(CultureInfo.CurrentCulture);
            }
            catch (DivideByZeroException)
            {
                MessageBox.Show("Cannot divide by zero.");
            }
        }

        // ===== CLEAR =====
        2 references
        private void btnClear_Click(object sender, EventArgs e)
        {
            txtNumber.Text = "0";
            lblResult.Text = "Result:";
            MoveCaretToEnd();
        }

        // ===== Helpers =====
        6 references
        private void MoveCaretToEnd()
        {
            txtNumber.SelectionStart = txtNumber.TextLength;
            txtNumber.Focus();
        }

        // Find first operator, ignoring a leading '-' (negative first number)
        2 references
        private int FindOperatorIndex(string expr)
        {
            if (string.IsNullOrWhiteSpace(expr)) return -1;

            for (int i = 0; i < expr.Length; i++)
            {
                char c = expr[i];
                if (Ops.Contains(c))
                {
                    if (i == 0 && c == '-') continue; // leading '-' is sign
                    return i;
                }
            }
            return -1;
        }

        2 references
        private bool TryParseDouble(string s, out double value)
        {
            return double.TryParse(
                s,
                NumberStyles.Float,
                CultureInfo.CurrentCulture,
                out value
            );
        }

        1 reference
        private double Compute(double a, double b, char op)
        {
            switch (op)
            {
                case '+': return a + b;
                case '-': return a - b;
                case '*': return a * b;
                case '/':
                    if (Math.Abs(b) < double.Epsilon) throw new DivideByZeroException();
                    return a / b;
                default: throw new InvalidOperationException("Unknown operator");
            }
        }
```

screenshots of running application



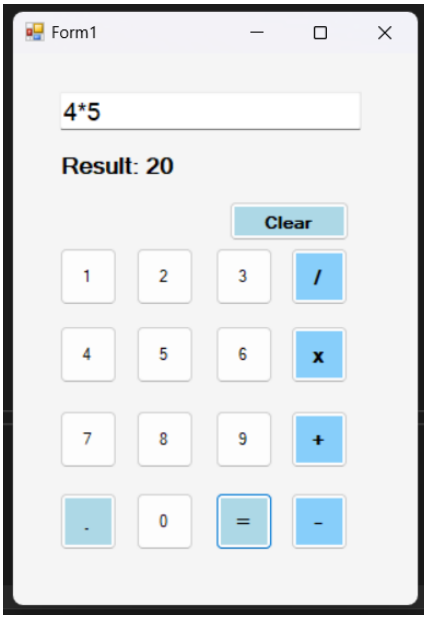
2+5

Result: 7



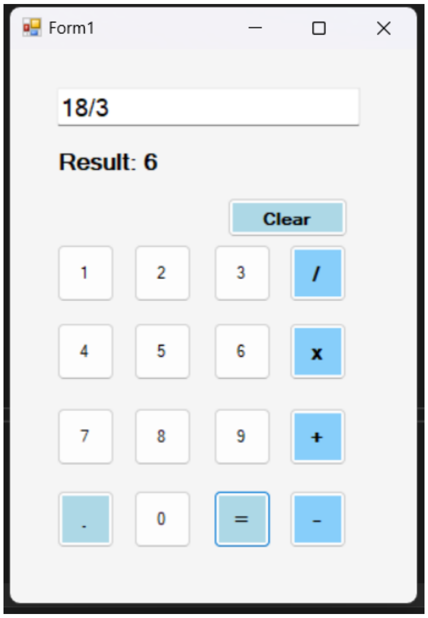
5-8

Result: -3



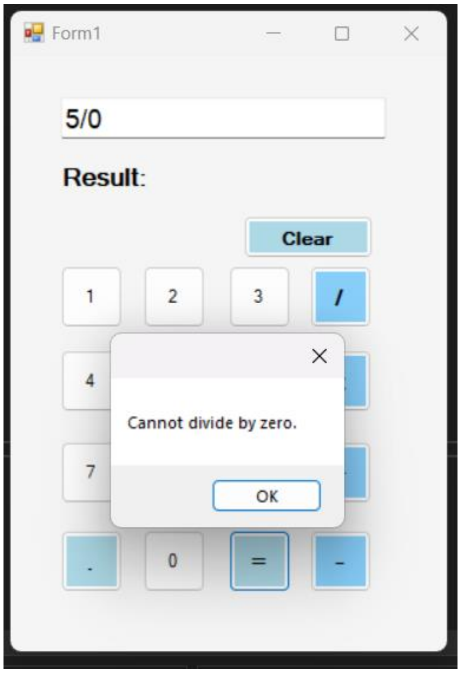
4*5

Result: 20



18/3

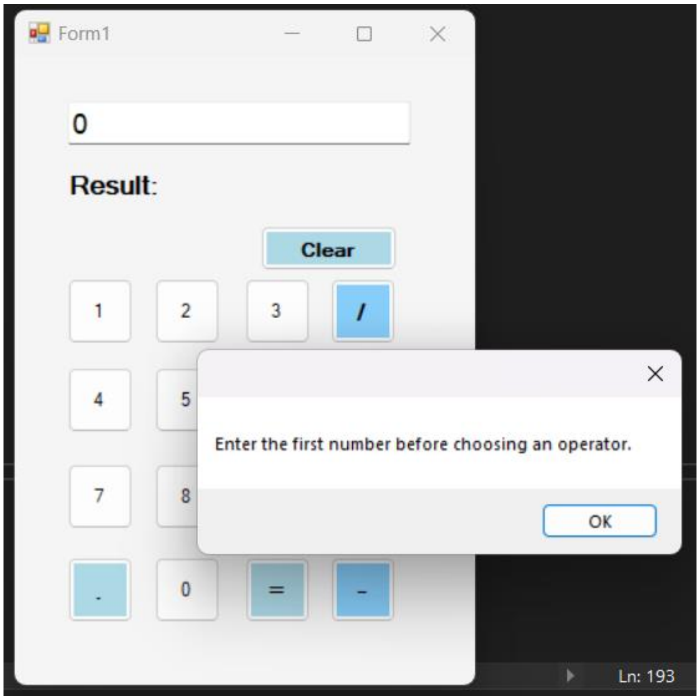
Result: 6



5/0

Result:

Cannot divide by zero.

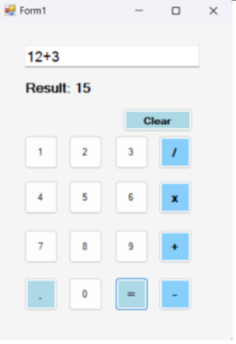
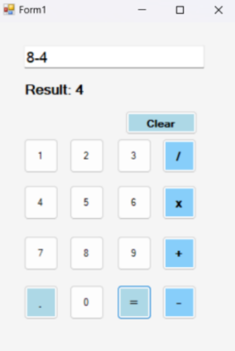
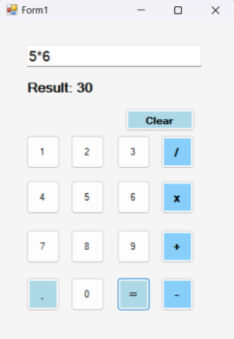


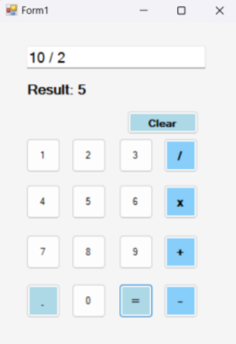
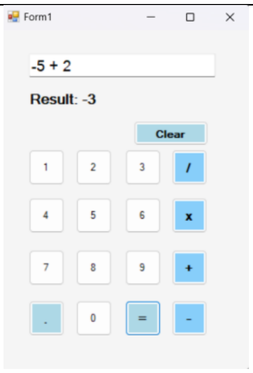
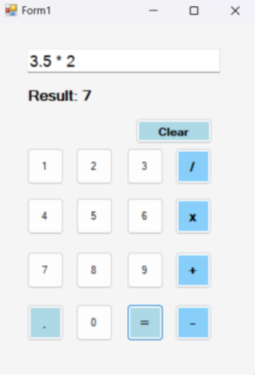
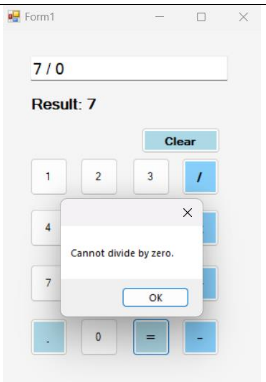
0

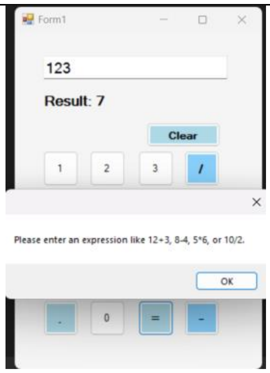
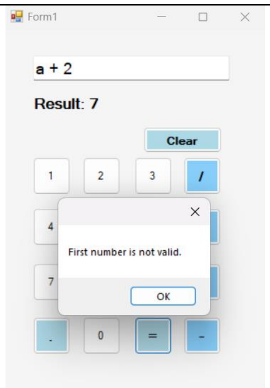
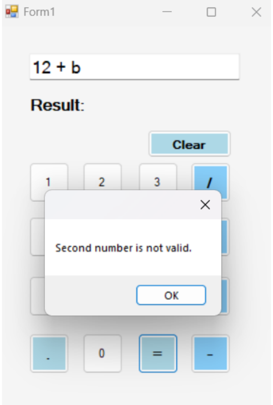
Result:

Enter the first number before choosing an operator.

Testing

Test Case ID	Component	Description	Data / Input	Expected Result	Actual Result	Pass / Fail
TC01	Addition	Verify correct addition of two positive numbers	12 + 3	Result: 15	 12+3 Result: 15 	**Pass**
TC02	Subtraction	Verify subtraction operation works correctly	8 - 4	Result: 4	 8-4 Result: 4 	**Pass**
TC03	Multiplication	Verify multiplication of two numbers	5 * 6	Result: 30	 5*6 Result: 30 	**Pass**

TC04	Division	Verify division gives correct result	10 / 2	Result: 5	 Form1 Clear 1 2 3 / 4 5 6 x 7 8 9 + . 0 = -10 / 2 Result: 5	**Pass**
TC05	Negative Numbers	Check addition with negative first number	-5 + 2	Result: -3	 Form1 Clear 1 2 3 / 4 5 6 x 7 8 9 + . 0 = --5 + 2 Result: -3	**Pass**
TC06	Decimal Numbers	Check multiplication with decimal input	3.5 * 2	Result: 7	 Form1 Clear 1 2 3 / 4 5 6 x 7 8 9 + . 0 = -3.5 * 2 Result: 7	**Pass**
TC07	Division by Zero	Check system response to divide by zero	7 / 0	Message: “Cannot divide by zero.”	 Form1 Clear 1 2 3 / 4 7 . 0 = - OK7 / 0 Result: 7 Cannot divide by zero.	**Pass**

TC08	Missing Operator	Verify validation when no operator used	123 =	Message: "Please enter an expression like 12+3..."	 123 Result: 7 Please enter an expression like 12+3, 8-4, 5*6, or 10/2. 	**Pass**
TC09	Invalid First Number	Validate input with non-numeric first value	a + 2	Message: "First number is not valid."	 a + 2 Result: 7 First number is not valid. 	**Pass**
TC10	Invalid Second Number	Validate input with non-numeric second value	12 + b	Message: "Second number is not valid."	 12 + b Result: Second number is not valid. 	**Pass**

Conclusion

The author successfully developed a functional Calculator Application using C# Windows Forms in Visual Studio. The system performs all basic arithmetic operations addition, subtraction, multiplication, and division accurately and efficiently. It also incorporates validation mechanisms to handle invalid inputs and division by zero, ensuring reliable and user-friendly performance. The interface is designed to be clear and simple, allowing smooth interaction between users and the application.

During the development process, the author gained practical experience in Graphical User Interface (GUI) design and event-driven programming using C#. The project enhanced understanding of how to connect user interface components, such as buttons and labels, with logical event handlers. Moreover, the author applied principles of input validation, exception handling, and software testing to ensure correctness and stability.

Overall, this project demonstrates the author’s ability to design, implement, and test a desktop application effectively, highlighting essential programming and problem-solving skills required for future software development tasks.