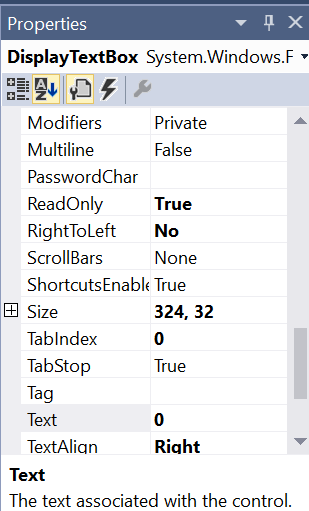
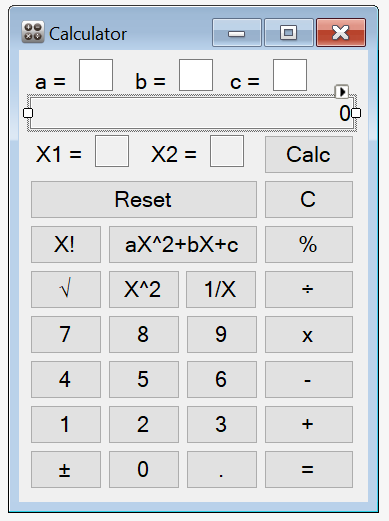
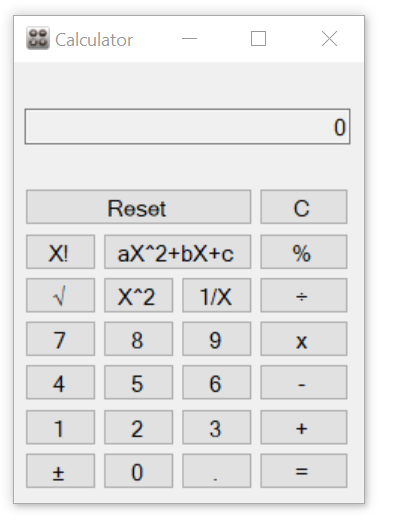
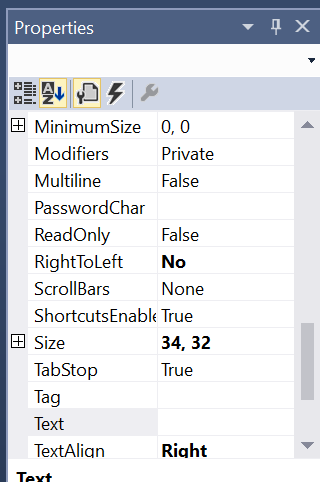
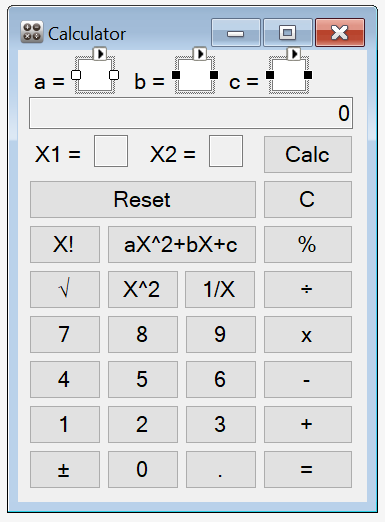
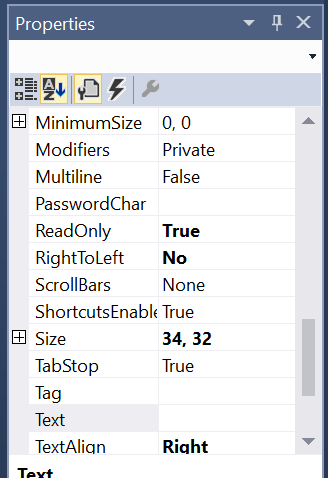
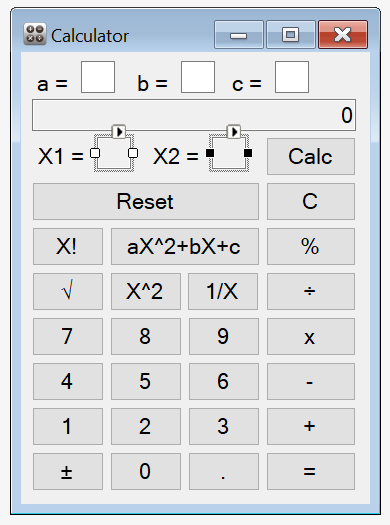
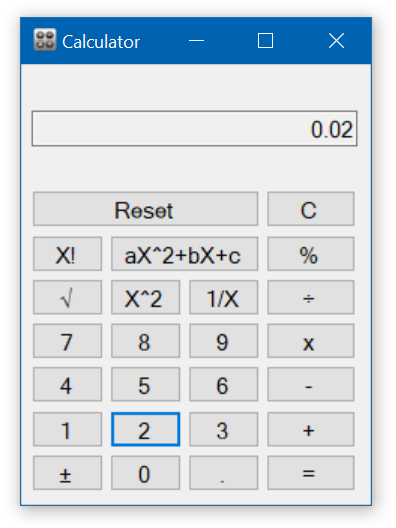
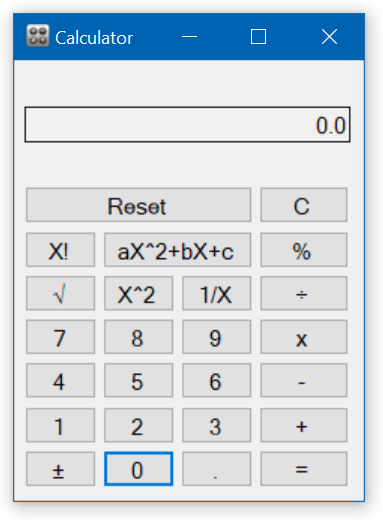
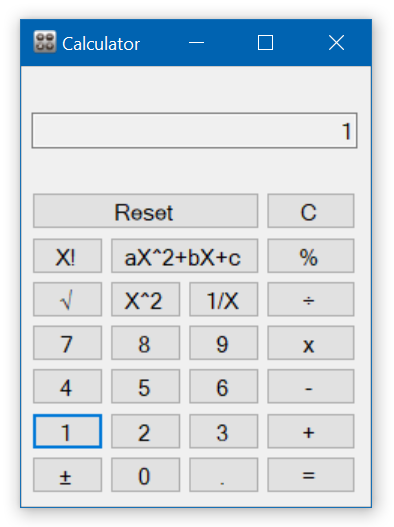
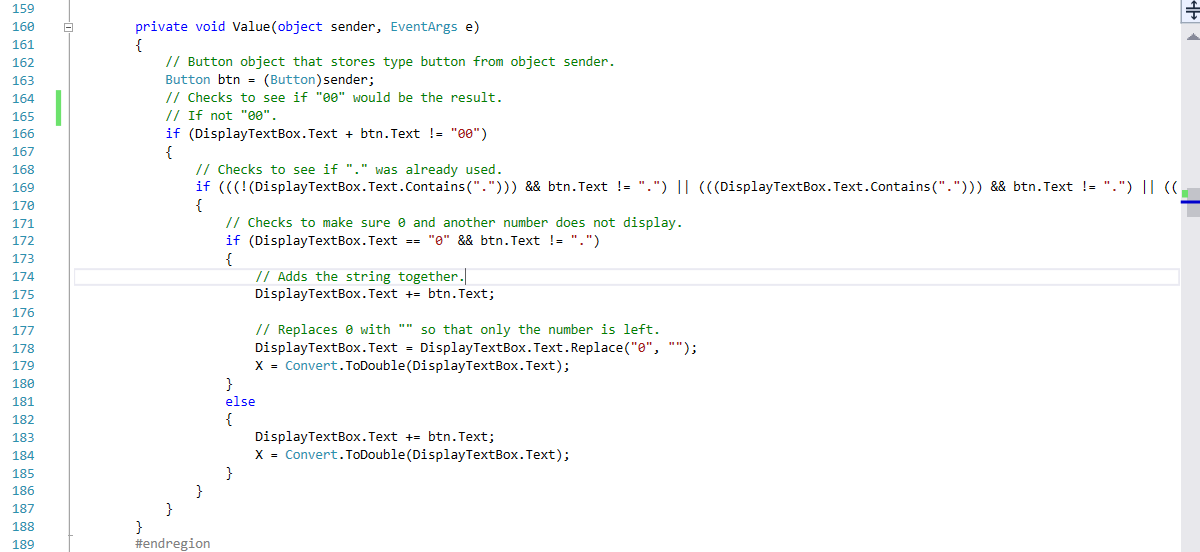
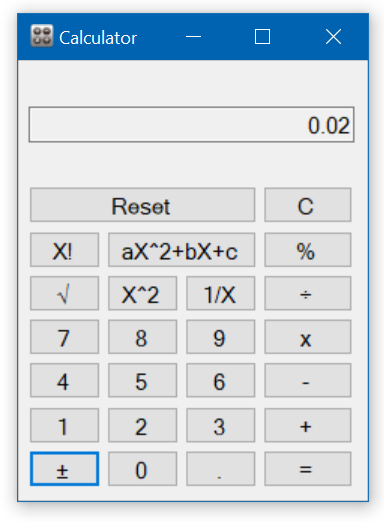
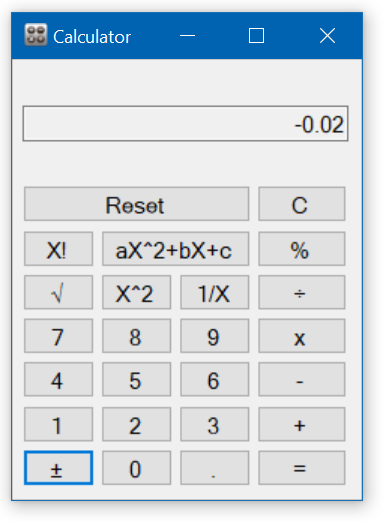
Calculator

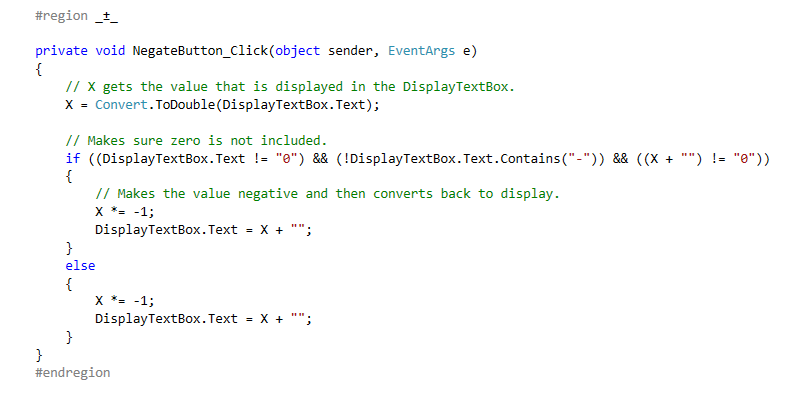


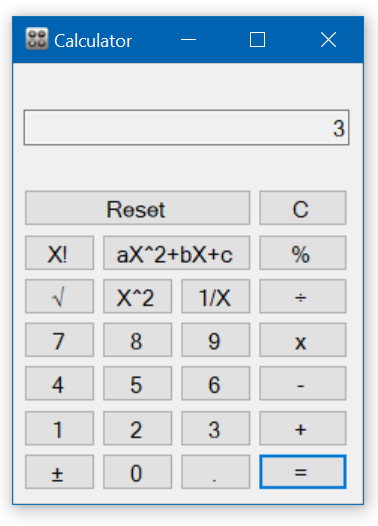
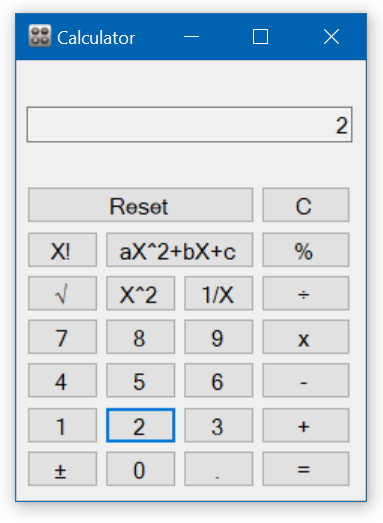
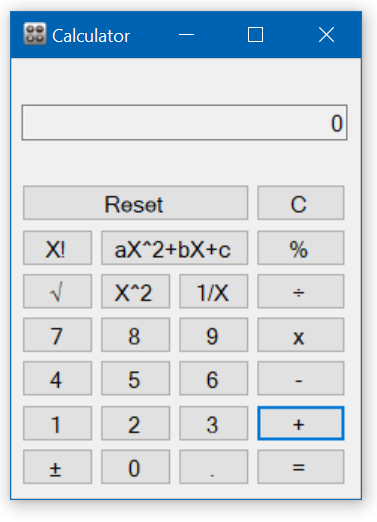
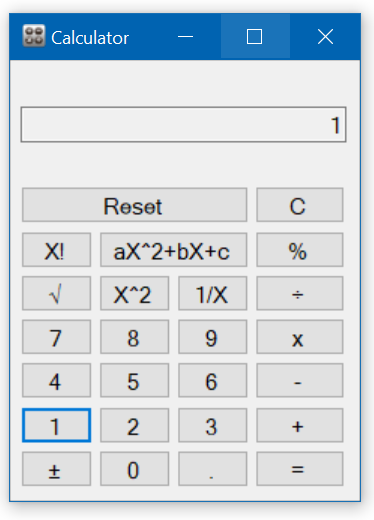


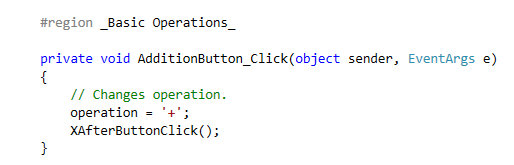


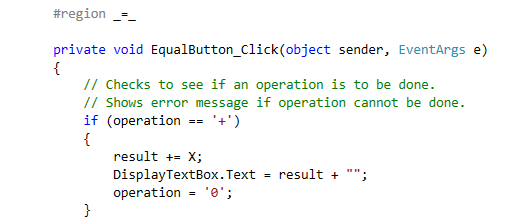


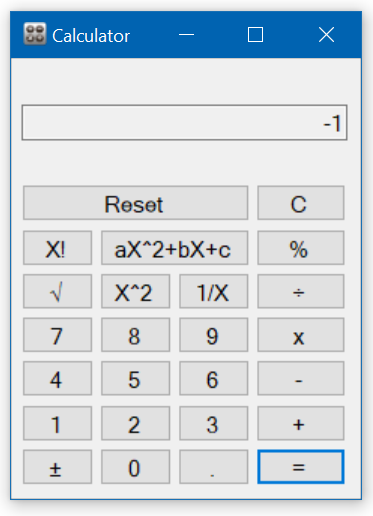
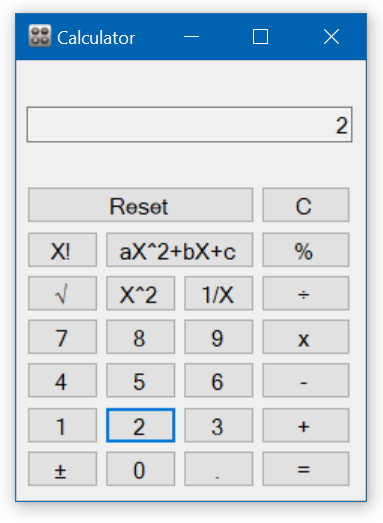
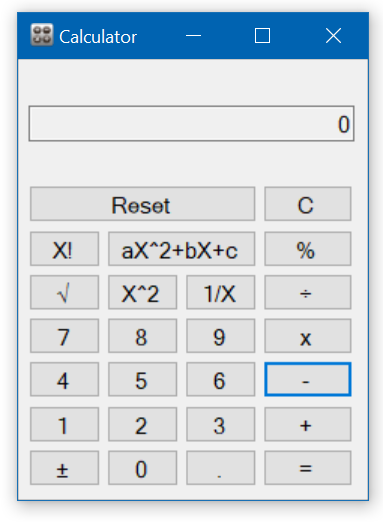
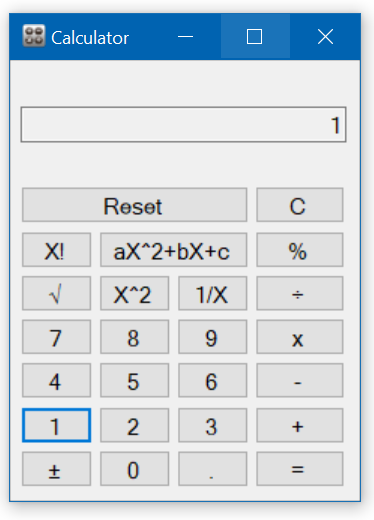


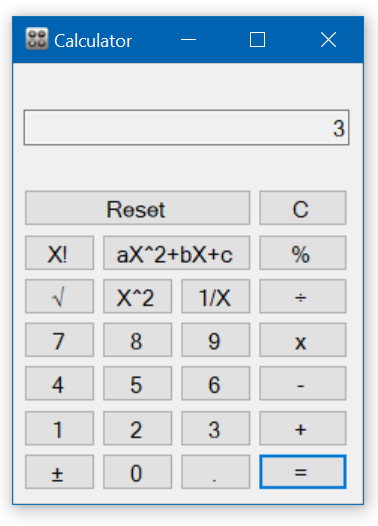
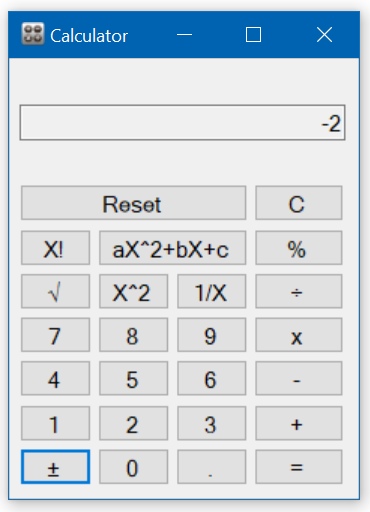
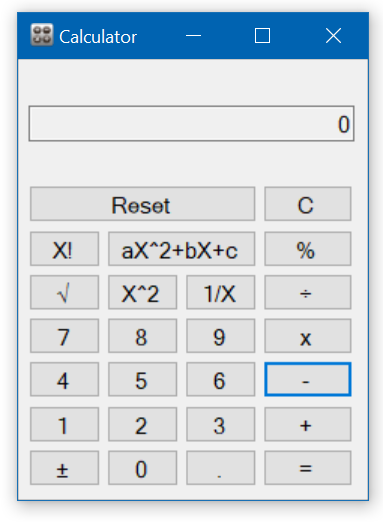
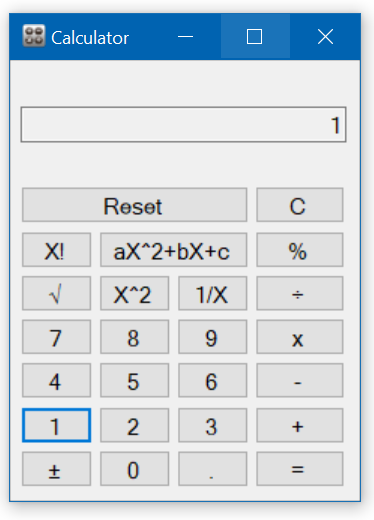


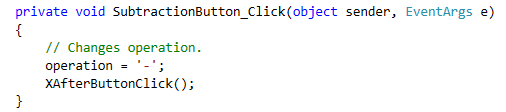


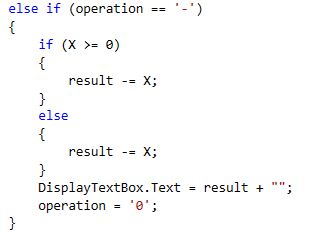


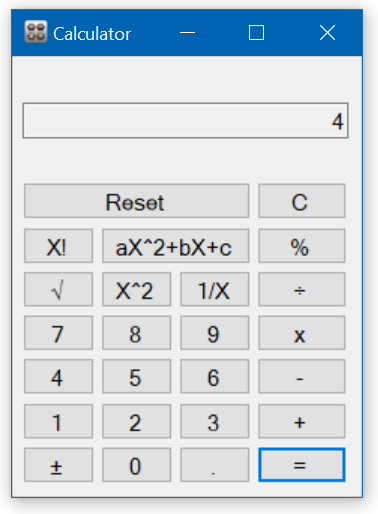
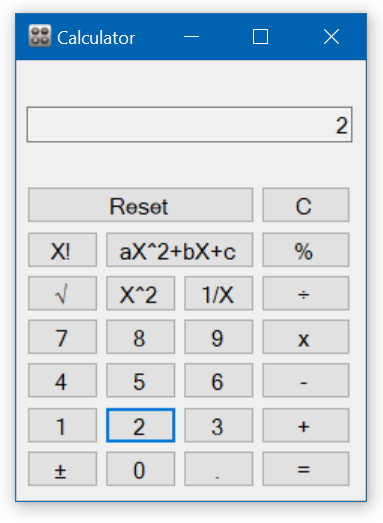
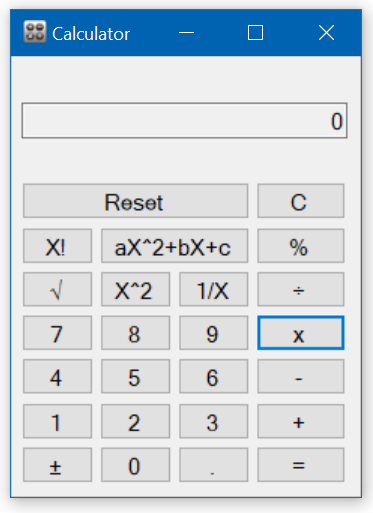
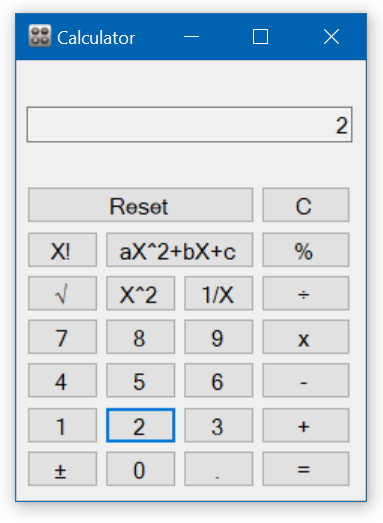


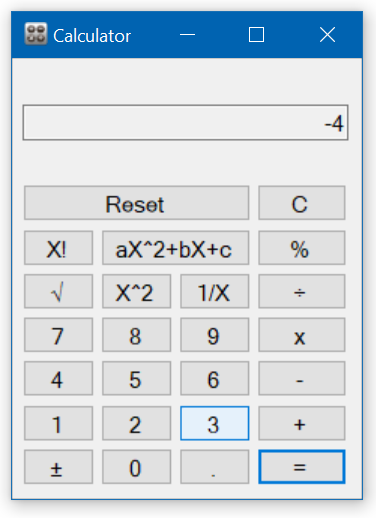
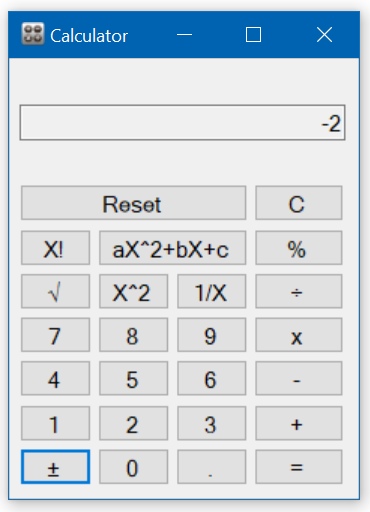
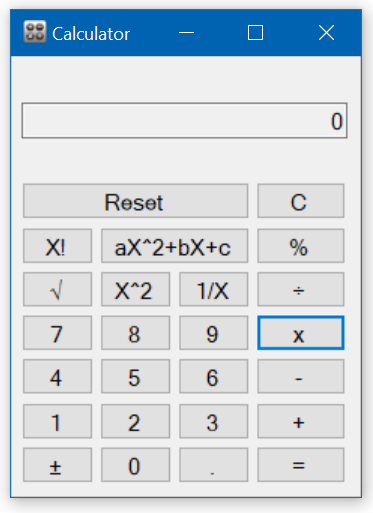
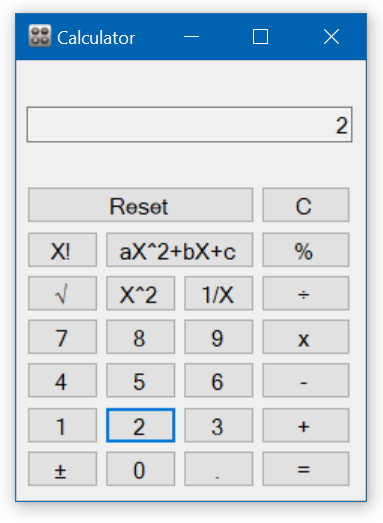


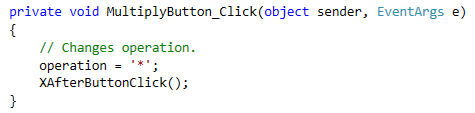


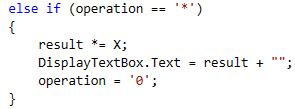


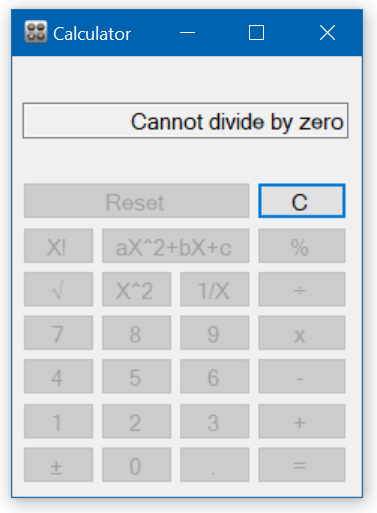
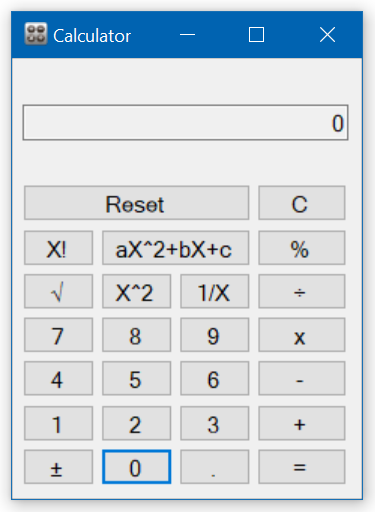
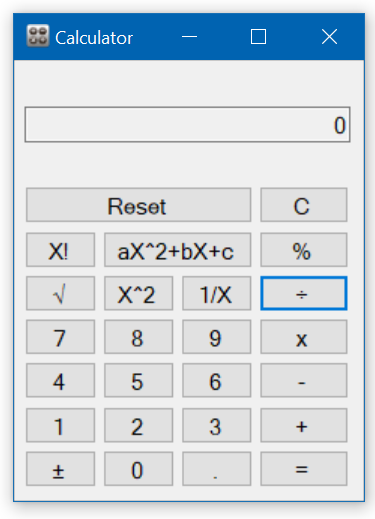
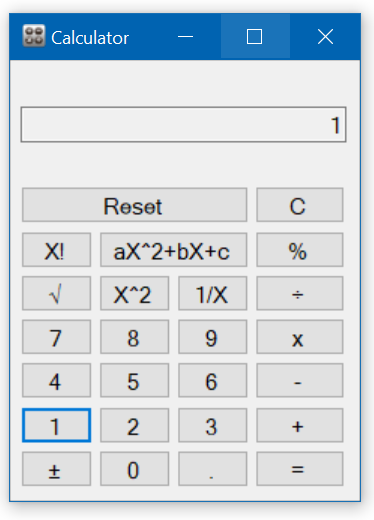


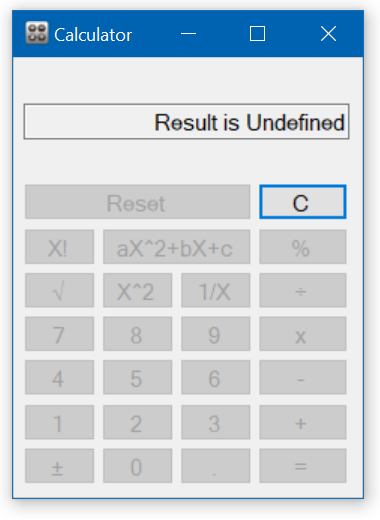
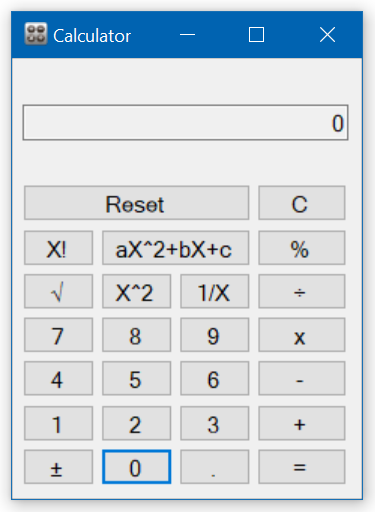
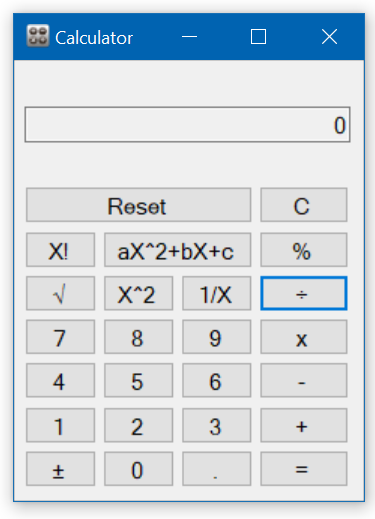
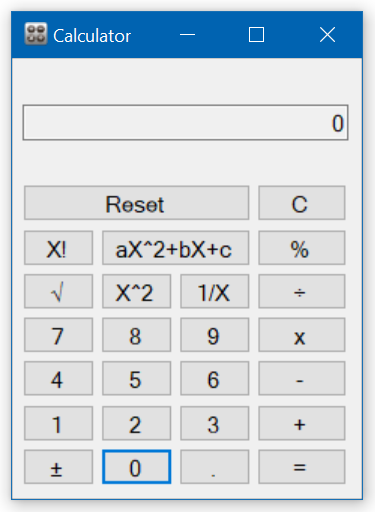


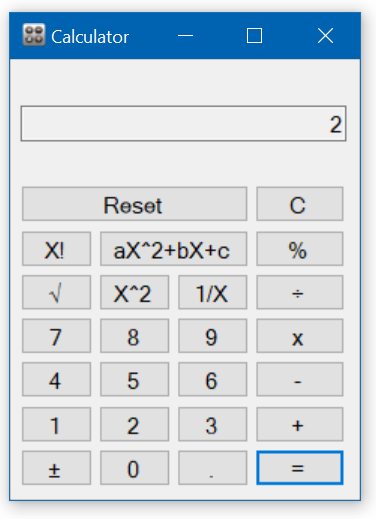
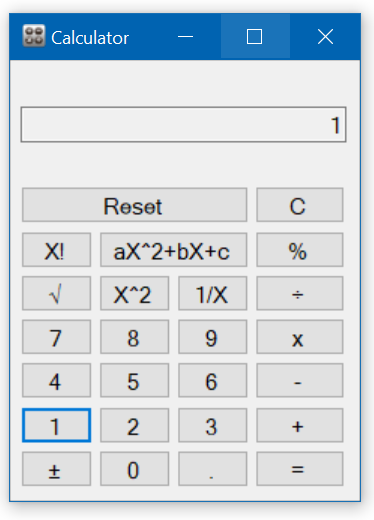
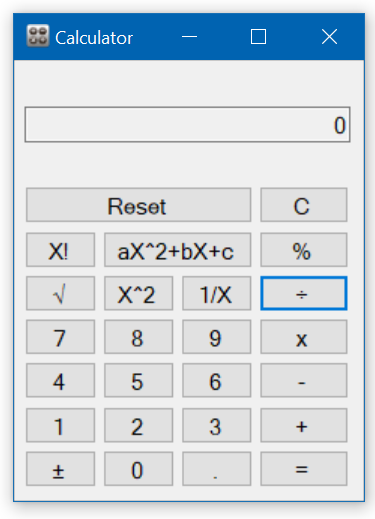
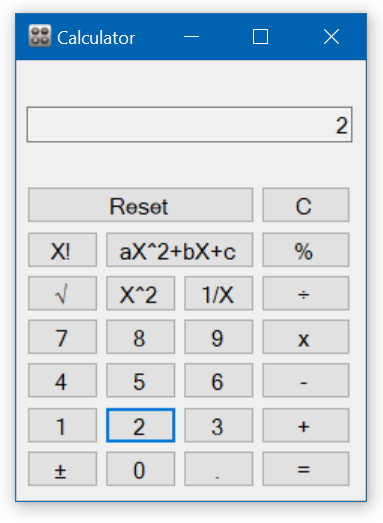


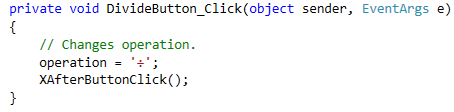


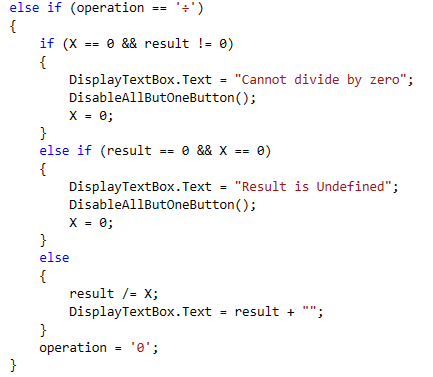


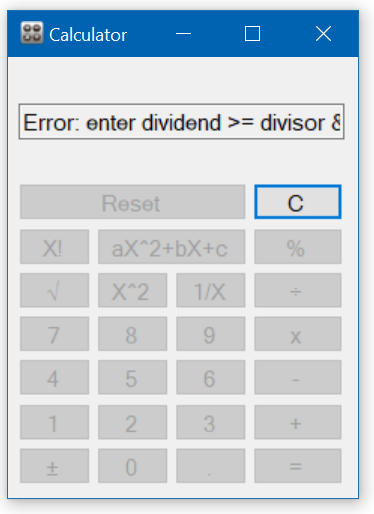
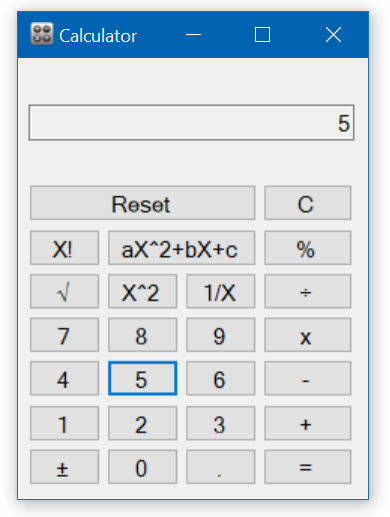
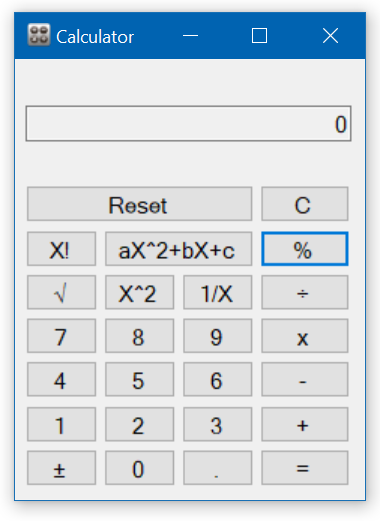
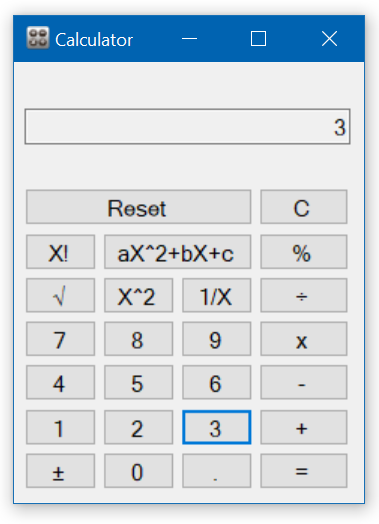
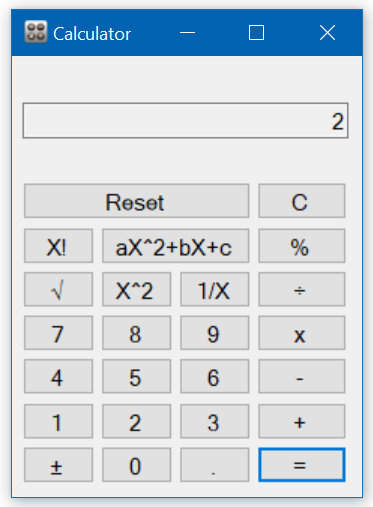
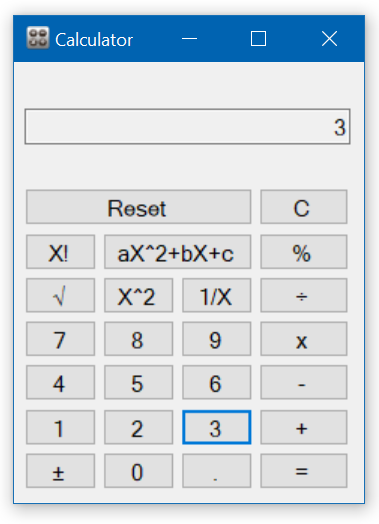
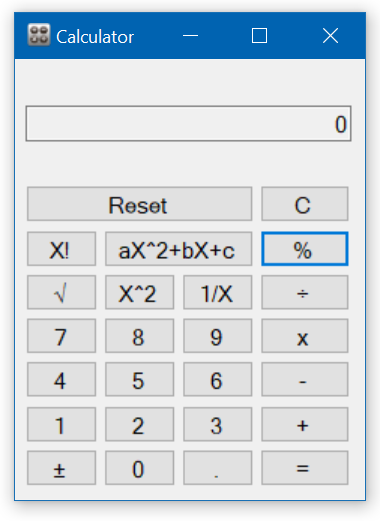
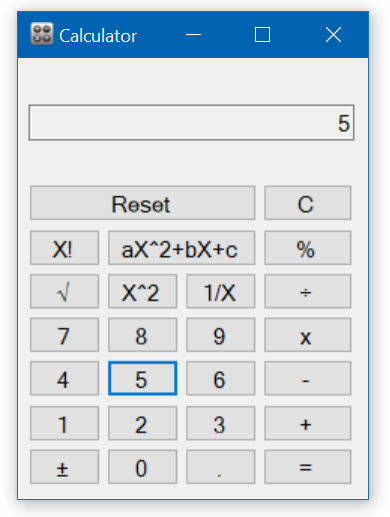


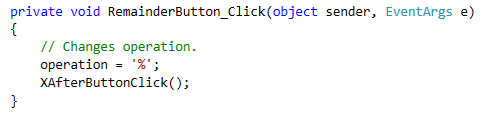


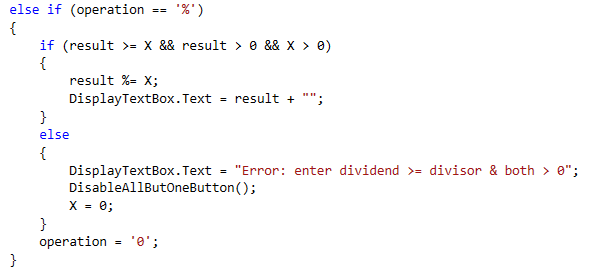


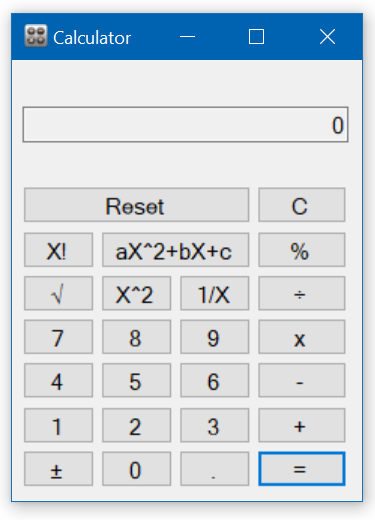
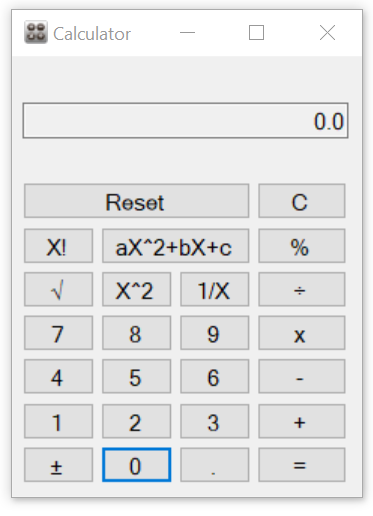


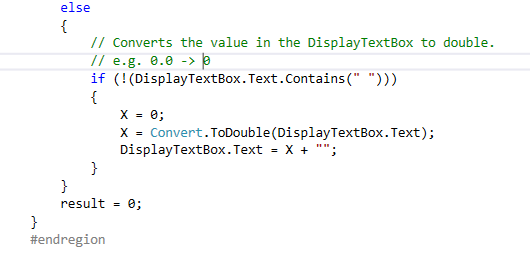


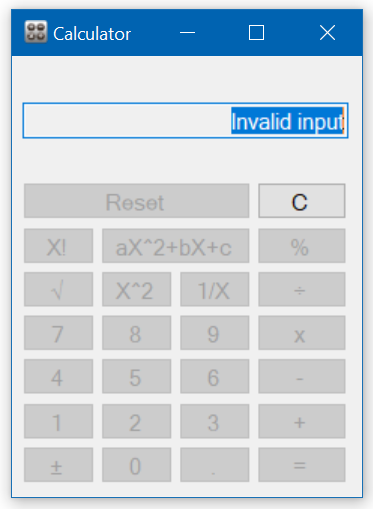
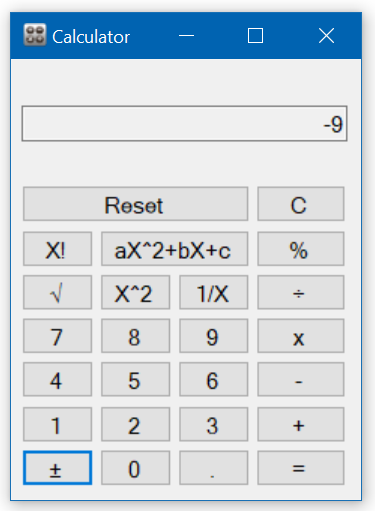


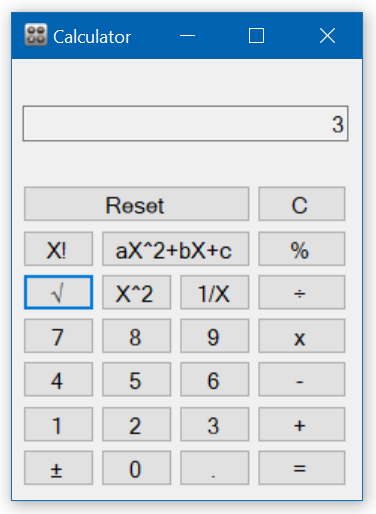
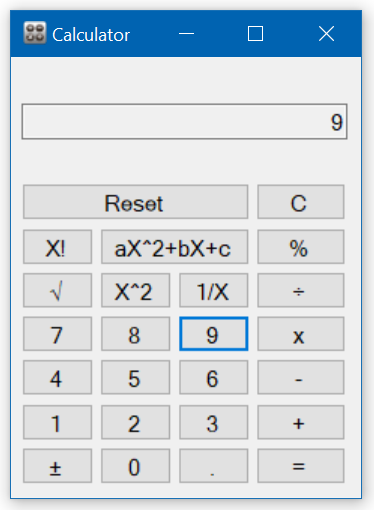


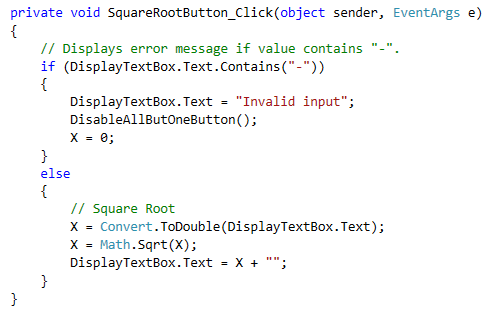


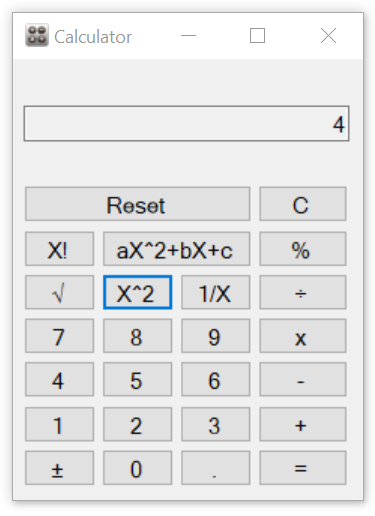
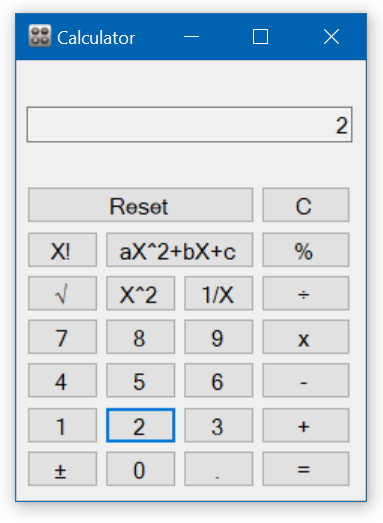


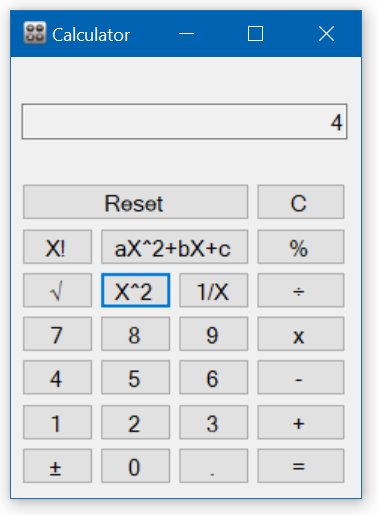
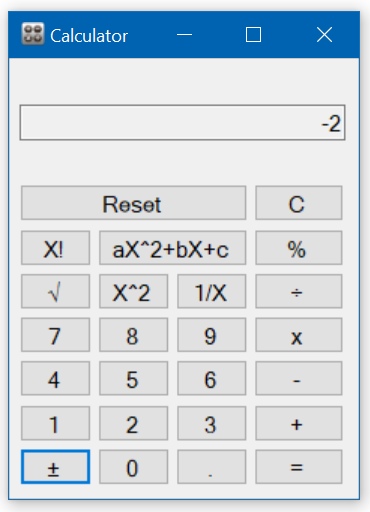


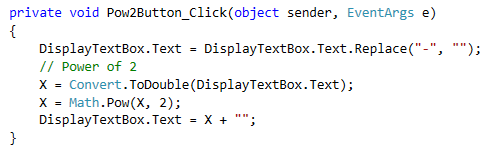


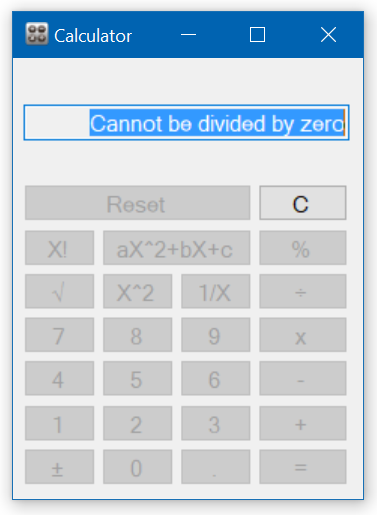
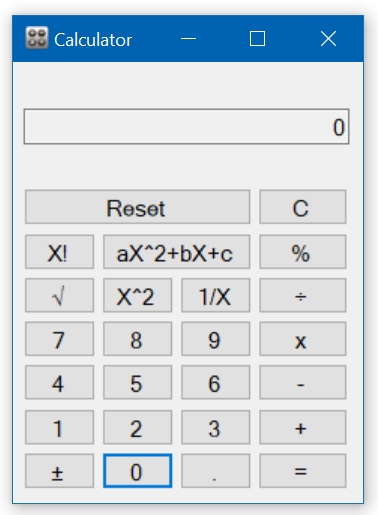


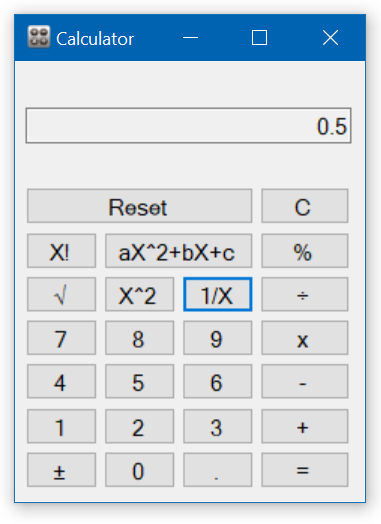
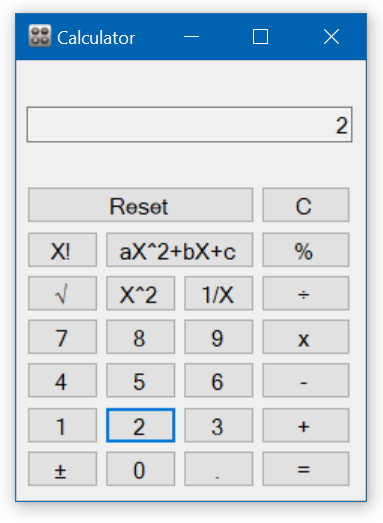


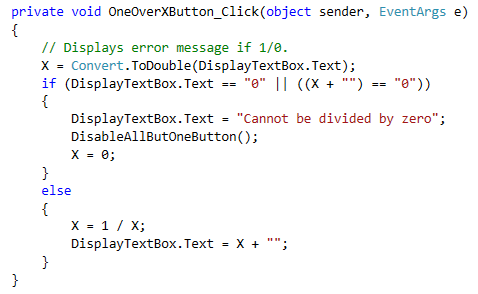


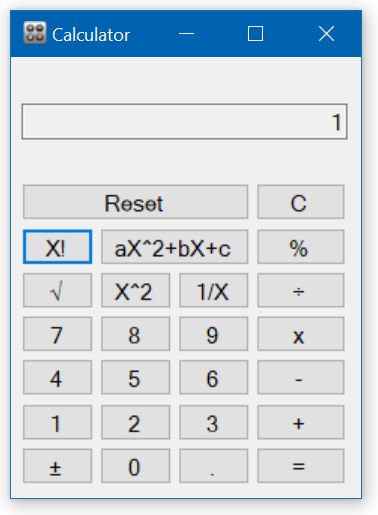
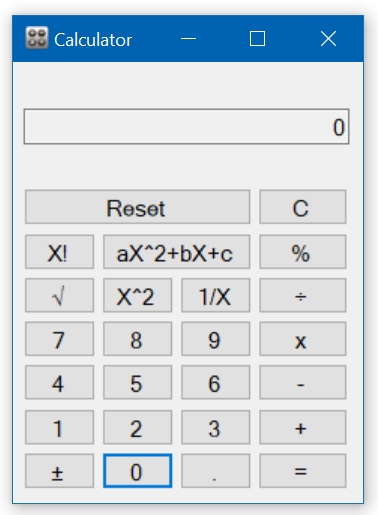


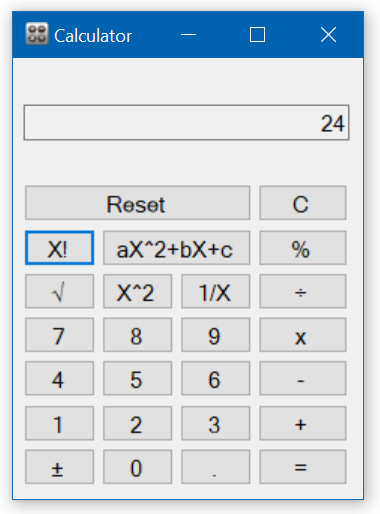
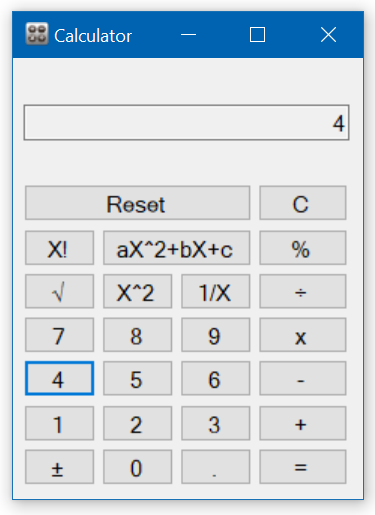


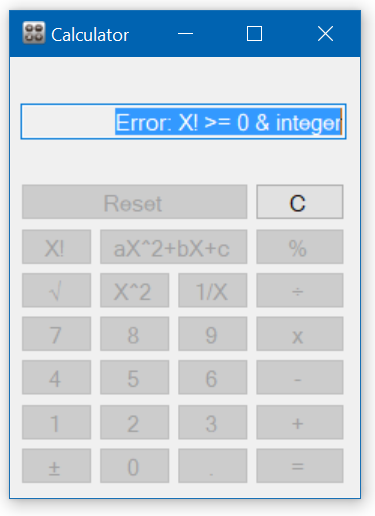
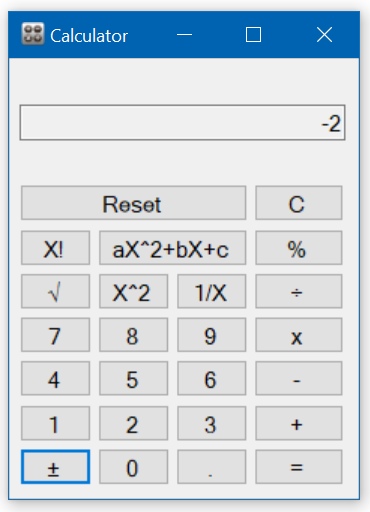


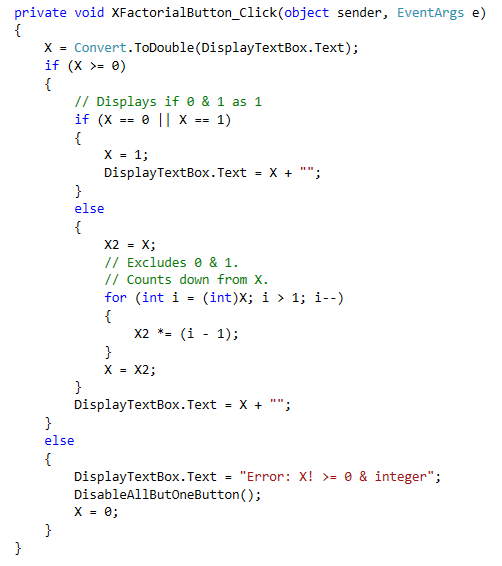


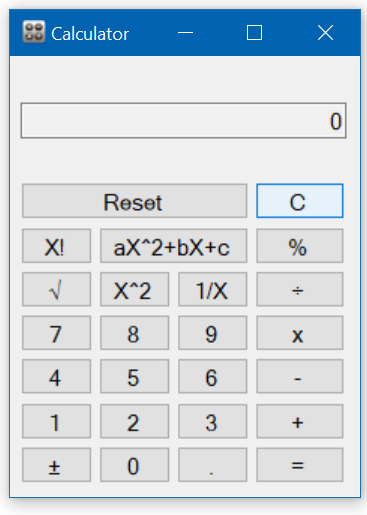
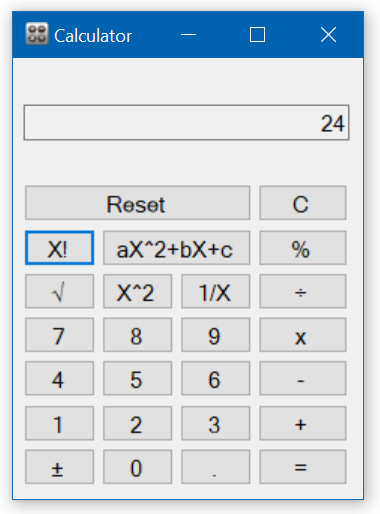
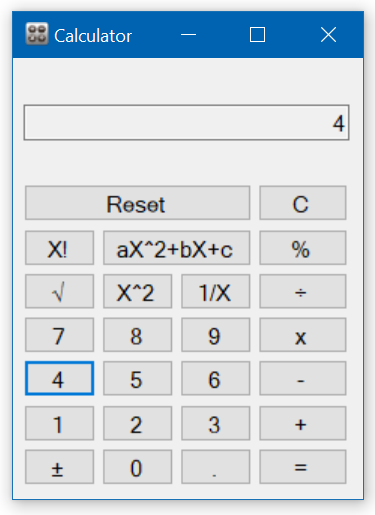


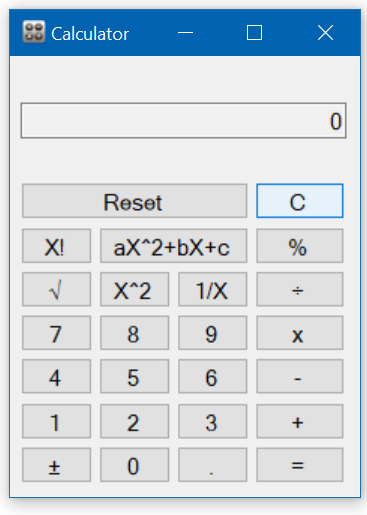
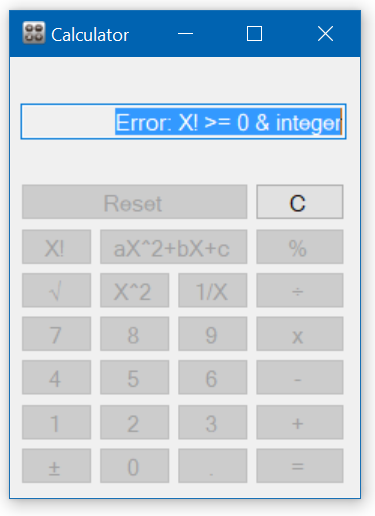
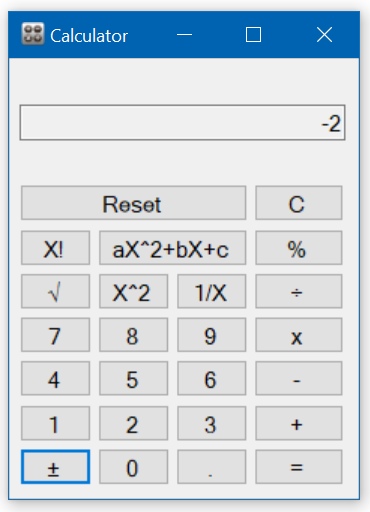


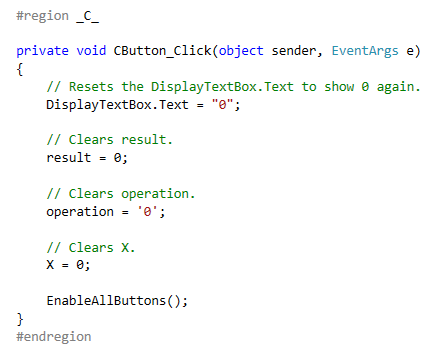


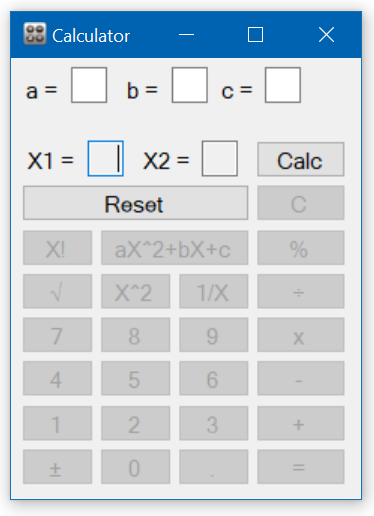


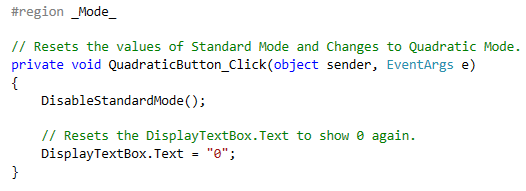


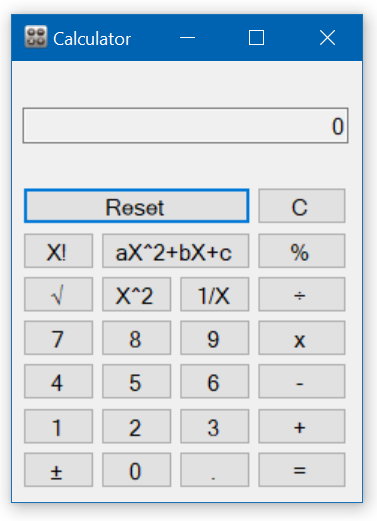
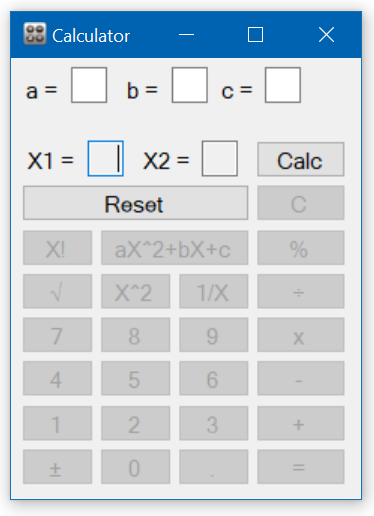


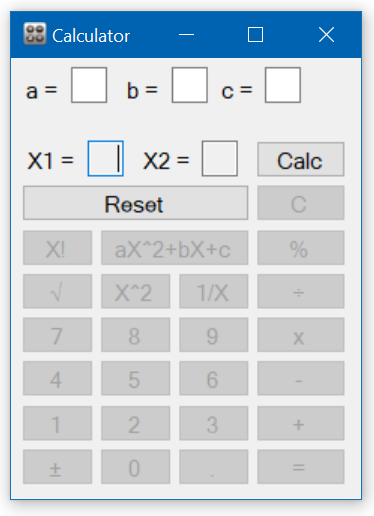
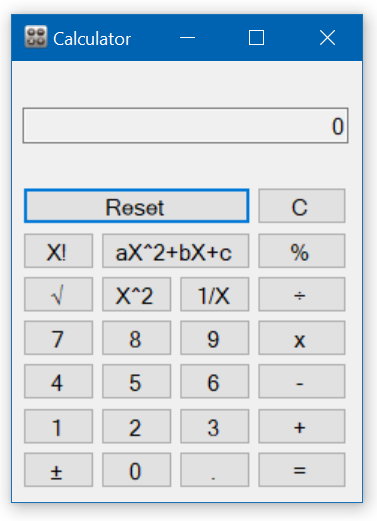
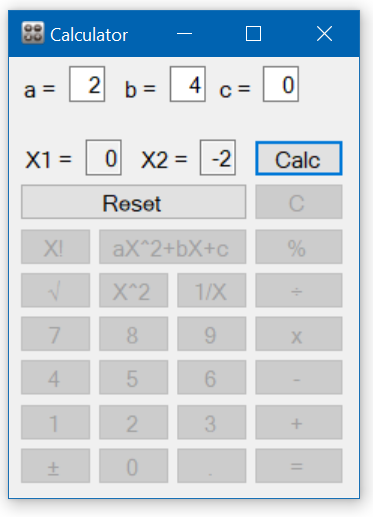


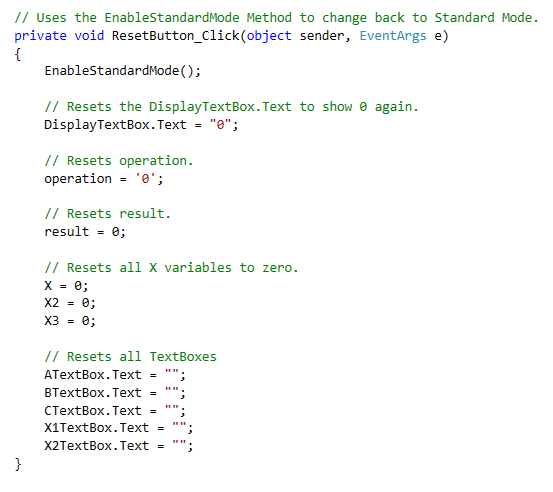


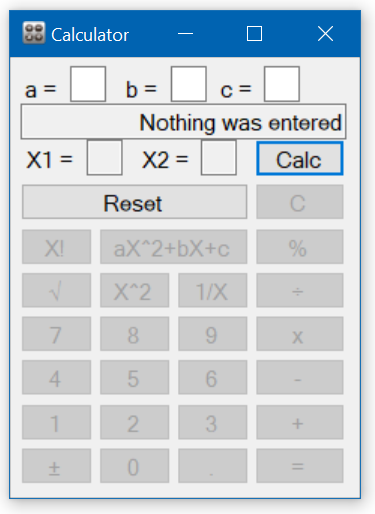
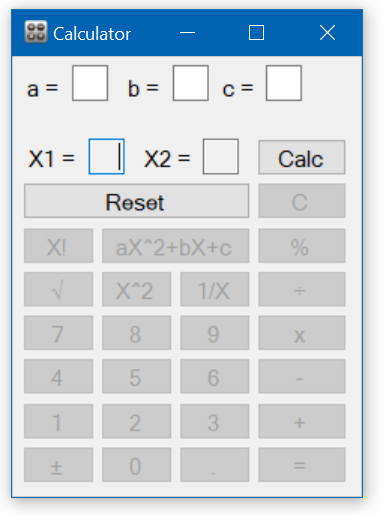


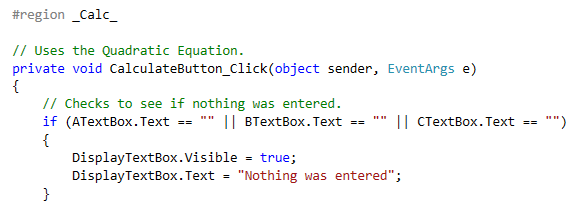


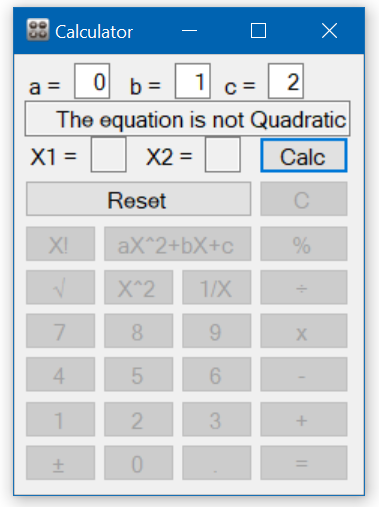
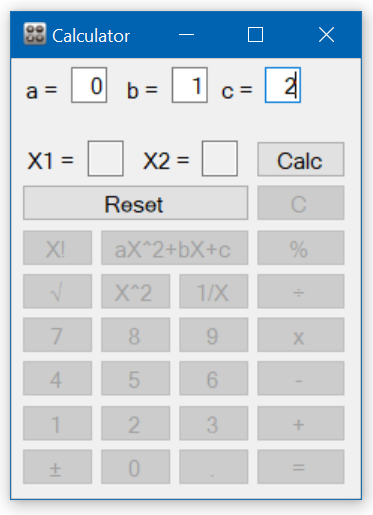


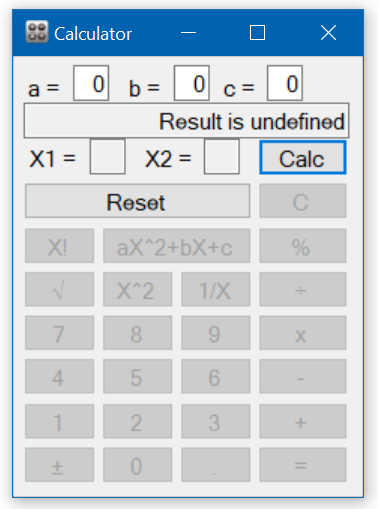
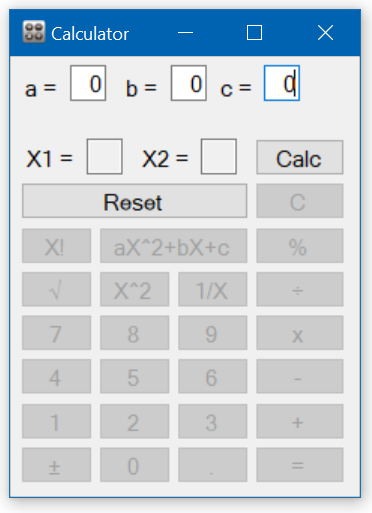


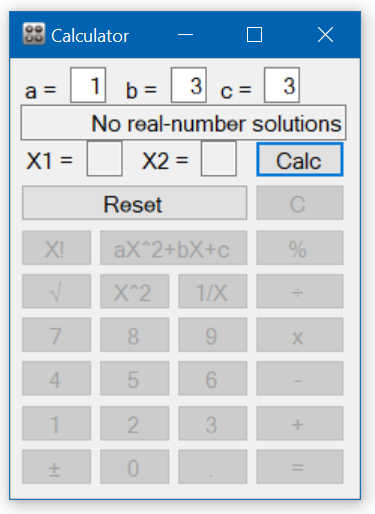
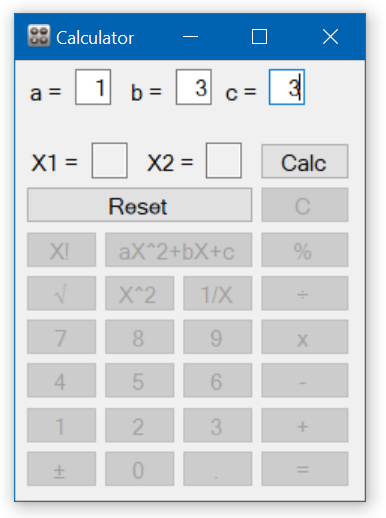


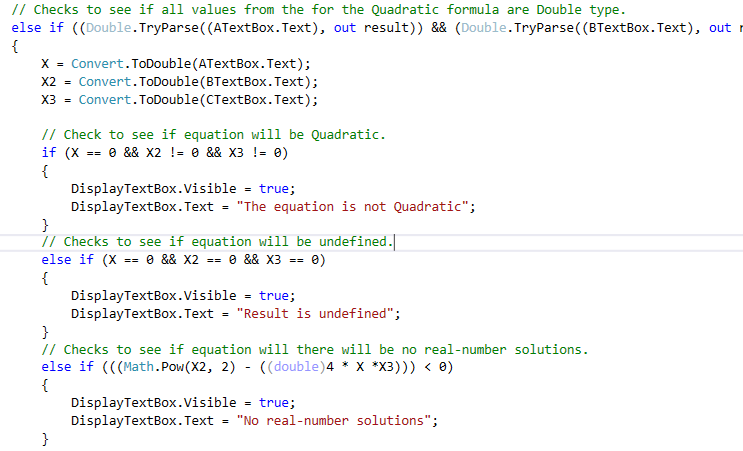


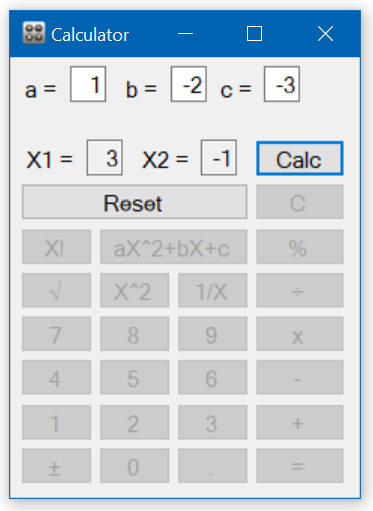
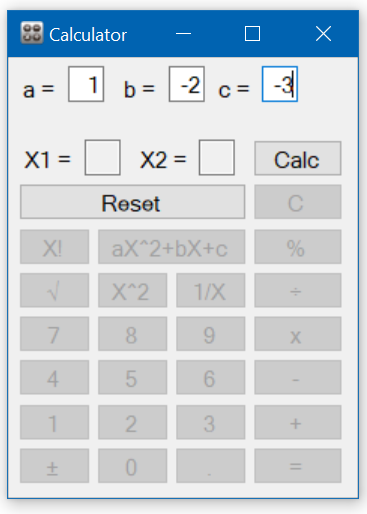


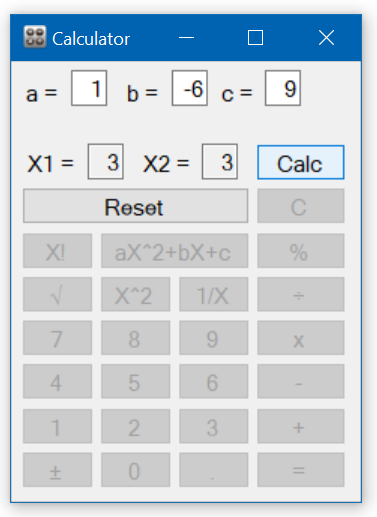
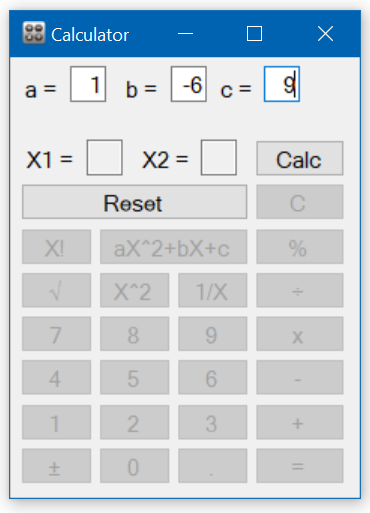


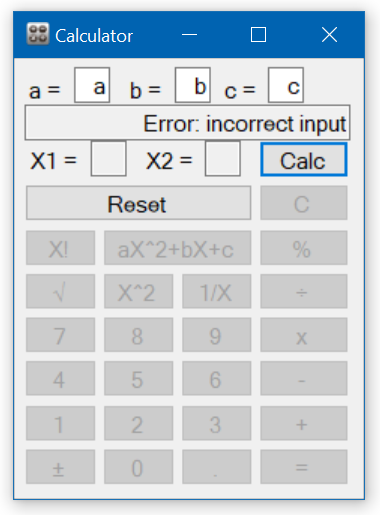
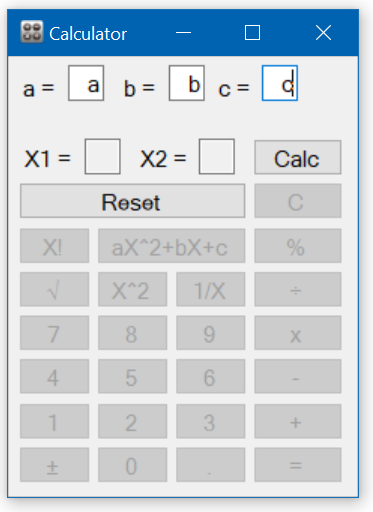


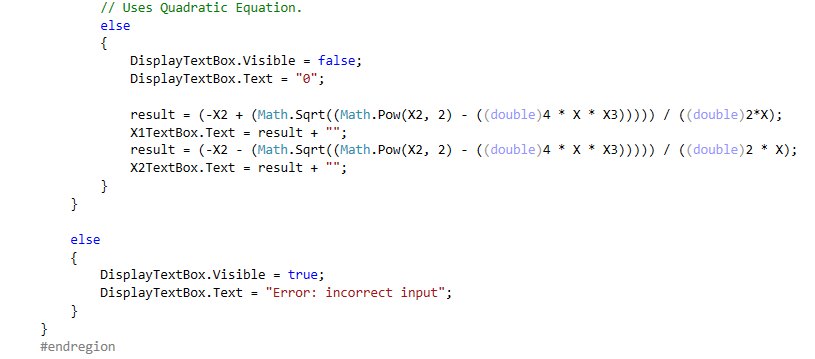












# Appendix

## CalculatorForm

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 Calculator

{

public partial class CalculatorForm : Form

{

private double X;

private double X2;

private double X3;

private double result;

private char operation;

public CalculatorForm()

{

InitializeComponent();

}

#region \_Additional Methods\_

#region \_X\_

private void XAfterButtonClick()

{

// Sets X to what is being diplayed.

X = Convert.ToDouble(DisplayTextBox.Text);

// Displays zero.

DisplayTextBox.Text = "0";

result += X;

X = 0;

}

#endregion

#region \_ Standard Calculator Error\_

// Disables all the buttons except for CButton.

private void DisableAllButOneButton()

{

EqualButton.Enabled = false;

AdditionButton.Enabled = false;

SubtractionButton.Enabled = false;

MultiplyButton.Enabled = false;

DivideButton.Enabled = false;

RemainderButton.Enabled = false;

ResetButton.Enabled = false;

XFactorialButton.Enabled = false;

QuadraticButton.Enabled = false;

SquareRootButton.Enabled = false;

Pow2Button.Enabled = false;

OneOverXButton.Enabled = false;

button15.Enabled = false;

button16.Enabled = false;

button17.Enabled = false;

button11.Enabled = false;

button18.Enabled = false;

button19.Enabled = false;

button20.Enabled = false;

button21.Enabled = false;

button22.Enabled = false;

NegateButton.Enabled = false;

button24.Enabled = false;

button25.Enabled = false;

CalculateButton.Enabled = false;

}

// Enables all the buttons except for CButton.

private void EnableAllButtons()

{

EqualButton.Enabled = true;

AdditionButton.Enabled = true;

SubtractionButton.Enabled = true;

MultiplyButton.Enabled = true;

DivideButton.Enabled = true;

RemainderButton.Enabled = true;

ResetButton.Enabled = true;

XFactorialButton.Enabled = true;

QuadraticButton.Enabled = true;

SquareRootButton.Enabled = true;

Pow2Button.Enabled = true;

OneOverXButton.Enabled = true;

button15.Enabled = true;

button16.Enabled = true;

button17.Enabled = true;

button11.Enabled = true;

button18.Enabled = true;

button19.Enabled = true;

button20.Enabled = true;

button21.Enabled = true;

button22.Enabled = true;

NegateButton.Enabled = true;

button24.Enabled = true;

button25.Enabled = true;

CalculateButton.Enabled = true;

}

#endregion

#region \_Mode Change\_

// Switches to Quadratic Mode.

private void DisableStandardMode()

{

ALabel.Visible = true;

BLabel.Visible = true;

CLabel.Visible = true;

X1Label.Visible = true;

X2Label.Visible = true;

ATextBox.Visible = true;

BTextBox.Visible = true;

CTextBox.Visible = true;

DisplayTextBox.Visible = false;

X1TextBox.Visible = true;

X2TextBox.Visible = true;

CalculateButton.Visible = true;

DisableAllButOneButton();

ResetButton.Enabled = true;

CalculateButton.Enabled = true;

CButton.Enabled = false;

}

// Switches back to Standard Mode.

private void EnableStandardMode()

{

ALabel.Visible = false;

BLabel.Visible = false;

CLabel.Visible = false;

X1Label.Visible = false;

X2Label.Visible = false;

ATextBox.Visible = false;

BTextBox.Visible = false;

CTextBox.Visible = false;

DisplayTextBox.Visible = true;

X1TextBox.Visible = false;

X2TextBox.Visible = false;

CalculateButton.Visible = false;

EnableAllButtons();

CButton.Enabled = true;

}

#endregion

#endregion

#region \_0 - 9 (includes '.')\_

private void Value(object sender, EventArgs e)

{

// Button object that stores type button from object sender.

Button btn = (Button)sender;

// Checks to see if "00" would be the result.

// If not "00".

if (DisplayTextBox.Text + btn.Text != "00")

{

// Checks to see if "." was already used.

if (((!(DisplayTextBox.Text.Contains("."))) && btn.Text != ".") || (((DisplayTextBox.Text.Contains("."))) && btn.Text != ".") || ((!(DisplayTextBox.Text.Contains("."))) && btn.Text == "."))

{

// Checks to make sure 0 and another number does not display.

if (DisplayTextBox.Text == "0" && btn.Text != ".")

{

// Adds the string together.

DisplayTextBox.Text += btn.Text;

// Replaces 0 with "" so that only the number is left.

DisplayTextBox.Text = DisplayTextBox.Text.Replace("0", "");

X = Convert.ToDouble(DisplayTextBox.Text);

}

else

{

DisplayTextBox.Text += btn.Text;

X = Convert.ToDouble(DisplayTextBox.Text);

}

}

}

}

#endregion

#region \_=\_

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

{

// Checks to see if an operation is to be done.

// Shows error message if operation cannot be done.

if (operation == '+')

{

result += X;

DisplayTextBox.Text = result + "";

operation = '0';

}

else if (operation == '-')

{

if (X >= 0)

{

result -= X;

}

else

{

result -= X;

}

DisplayTextBox.Text = result + "";

operation = '0';

}

else if (operation == '\*')

{

result \*= X;

DisplayTextBox.Text = result + "";

operation = '0';

}

else if (operation == '÷')

{

if (X == 0 && result != 0)

{

DisplayTextBox.Text = "Cannot divide by zero";

DisableAllButOneButton();

X = 0;

}

else if (result == 0 && X == 0)

{

DisplayTextBox.Text = "Result is Undefined";

DisableAllButOneButton();

X = 0;

}

else

{

result /= X;

DisplayTextBox.Text = result + "";

}

operation = '0';

}

else if (operation == '%')

{

if (result >= X && result > 0 && X > 0)

{

result %= X;

DisplayTextBox.Text = result + "";

}

else

{

DisplayTextBox.Text = "Error: enter dividend >= divisor & both > 0";

DisableAllButOneButton();

X = 0;

}

operation = '0';

}

else

{

// Converts the value in the DisplayTextBox to double.

// e.g. 0.0 -> 0

if (!(DisplayTextBox.Text.Contains(" ")))

{

X = 0;

X = Convert.ToDouble(DisplayTextBox.Text);

DisplayTextBox.Text = X + "";

}

}

result = 0;

}

#endregion

#region \_±\_

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

{

// X gets the value that is displayed in the DisplayTextBox.

X = Convert.ToDouble(DisplayTextBox.Text);

// Makes sure zero is not included.

if ((DisplayTextBox.Text != "0") && (!DisplayTextBox.Text.Contains("-")) && ((X + "") != "0"))

{

// Makes the value negative and then converts back to display.

X \*= -1;

DisplayTextBox.Text = X + "";

}

else

{

X \*= -1;

DisplayTextBox.Text = X + "";

}

}

#endregion

#region \_Basic Operations\_

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

{

// Changes operation.

operation = '+';

XAfterButtonClick();

}

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

{

// Changes operation.

operation = '-';

XAfterButtonClick();

}

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

{

// Changes operation.

operation = '\*';

XAfterButtonClick();

}

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

{

// Changes operation.

operation = '÷';

XAfterButtonClick();

}

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

{

// Changes operation.

operation = '%';

XAfterButtonClick();

}

#endregion

#region \_Advanced Operations\_

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

{

// Displays error message if value contains "-".

if (DisplayTextBox.Text.Contains("-"))

{

DisplayTextBox.Text = "Invalid input";

DisableAllButOneButton();

X = 0;

}

else

{

// Square Root

X = Convert.ToDouble(DisplayTextBox.Text);

X = Math.Sqrt(X);

DisplayTextBox.Text = X + "";

}

}

private void Pow2Button\_Click(object sender, EventArgs e)

{

DisplayTextBox.Text = DisplayTextBox.Text.Replace("-", "");

// Power of 2

X = Convert.ToDouble(DisplayTextBox.Text);

X = Math.Pow(X, 2);

DisplayTextBox.Text = X + "";

}

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

{

// Displays error message if 1/0.

X = Convert.ToDouble(DisplayTextBox.Text);

if (DisplayTextBox.Text == "0" || ((X + "") == "0"))

{

DisplayTextBox.Text = "Cannot be divided by zero";

DisableAllButOneButton();

X = 0;

}

else

{

X = 1 / X;

DisplayTextBox.Text = X + "";

}

}

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

{

X = Convert.ToDouble(DisplayTextBox.Text);

if (X >= 0)

{

// Displays if 0 & 1 as 1

if (X == 0 || X == 1)

{

X = 1;

DisplayTextBox.Text = X + "";

}

else

{

X2 = X;

// Excludes 0 & 1.

// Counts down from X.

for (int i = (int)X; i > 1; i--)

{

X2 \*= (i - 1);

}

X = X2;

}

DisplayTextBox.Text = X + "";

}

else

{

DisplayTextBox.Text = "Error: X! >= 0 & integer";

DisableAllButOneButton();

X = 0;

}

}

#endregion

#region \_C\_

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

{

// Resets the DisplayTextBox.Text to show 0 again.

DisplayTextBox.Text = "0";

// Clears result.

result = 0;

// Clears operation.

operation = '0';

// Clears X.

X = 0;

EnableAllButtons();

}

#endregion

#region \_Mode\_

// Resets the values of Standard Mode and Changes to Quadratic Mode.

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

{

DisableStandardMode();

// Resets the DisplayTextBox.Text to show 0 again.

DisplayTextBox.Text = "0";

}

// Uses the EnableStandardMode Method to change back to Standard Mode.

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

{

EnableStandardMode();

// Resets the DisplayTextBox.Text to show 0 again.

DisplayTextBox.Text = "0";

// Resets operation.

operation = '0';

// Resets result.

result = 0;

// Resets all X variables to zero.

X = 0;

X2 = 0;

X3 = 0;

// Resets all TextBoxes

ATextBox.Text = "";

BTextBox.Text = "";

CTextBox.Text = "";

X1TextBox.Text = "";

X2TextBox.Text = "";

}

#endregion

#region \_Calc\_

// Uses the Quadratic Equation.

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

{

// Checks to see if nothing was entered.

if (ATextBox.Text == "" || BTextBox.Text == "" || CTextBox.Text == "")

{

DisplayTextBox.Visible = true;

DisplayTextBox.Text = "Nothing was entered";

}

// Checks to see if all values from the for the Quadratic formula are Double type.

else if ((Double.TryParse((ATextBox.Text), out result)) && (Double.TryParse((BTextBox.Text), out result)) && (Double.TryParse((ATextBox.Text), out result)))

{

X = Convert.ToDouble(ATextBox.Text);

X2 = Convert.ToDouble(BTextBox.Text);

X3 = Convert.ToDouble(CTextBox.Text);

// Check to see if equation will be Quadratic.

if (X == 0 && X2 != 0 && X3 != 0)

{

DisplayTextBox.Visible = true;

DisplayTextBox.Text = "The equation is not Quadratic";

}

// Checks to see if equation will be undefined.

else if (X == 0 && X2 == 0 && X3 == 0)

{

DisplayTextBox.Visible = true;

DisplayTextBox.Text = "Result is undefined";

}

// Checks to see if equation will there will be no real-number solutions.

else if (((Math.Pow(X2, 2) - ((double)4 \* X \*X3))) < 0)

{

DisplayTextBox.Visible = true;

DisplayTextBox.Text = "No real-number solutions";

}

// Uses Quadratic Equation.

else

{

DisplayTextBox.Visible = false;

DisplayTextBox.Text = "0";

result = (-X2 + (Math.Sqrt((Math.Pow(X2, 2) - ((double)4 \* X \* X3))))) / ((double)2\*X);

X1TextBox.Text = result + "";

result = (-X2 - (Math.Sqrt((Math.Pow(X2, 2) - ((double)4 \* X \* X3))))) / ((double)2 \* X);

X2TextBox.Text = result + "";

}

}

else

{

DisplayTextBox.Visible = true;

DisplayTextBox.Text = "Error: incorrect input";

}

}

#endregion

}

}