

Calculator Technical Report

By James Richards | 06/08/19

https://github.com/MahoganyBlue/Calculator.git

Table of Contents

[Data Structures 1](#_Toc11154935)

[Algorithms 2](#_Toc11154936)

[ Number Algorithms 2](#_Toc11154937)

[ Clear Button 2](#_Toc11154938)

[ Dot button 2](#_Toc11154939)

[ Set FirstValue 2](#_Toc11154940)

[ btnEquals 3](#_Toc11154941)

[ Arithmetic btn 4](#_Toc11154942)

[ Trigonometric btn 4](#_Toc11154943)

[ Algebraic btn 4](#_Toc11154944)

[Cube Root btn 5](#_Toc11154945)

[Square Root btn 5](#_Toc11154946)

[Inverse btn 6](#_Toc11154947)

[ Square Root btn 6](#_Toc11154948)

[Testing Procedure 7](#_Toc11154949)

[Future Upgrades 7](#_Toc11154950)

# Data Structures

The data Structures are used to store the first and second value entered as well as which calculation the user selected to use (divide, plus, multiply).

|  |  |  |
| --- | --- | --- |
| Name | Type | Purpose |
| firstValue | Double | This variable takes the first value entered from the calculator and saves it for the calculation to come. |
| secondValue | Double | The secondValue variable is used to store the second value entered into the calculator after a calculation is picked EG. Plus, minus. This is used in the final calculation with the firstValue variable. |
| plusButtonClicked | Boolean | This variable holds the Boolean for the plus button. This lets the btnEquals\_Click method know which calculation to do. |
| minusButtonClicked | Boolean | This variable holds the Boolean for the minus button. This lets the btnEquals\_Click method know which calculation to do. |
| divideButtonClicked | Boolean | This variable holds the Boolean for the divide button. This lets the btnEquals\_Click method know which calculation to do. |
| multiplyButtonClicked | Boolean | This variable holds the Boolean for the multiply button. This lets the btnEquals\_Click method know which calculation to do. |
| TanButtonClicked | Boolean | This variable holds the Boolean for the Tan button. This lets the btnEquals\_Click method know which calculation to do. |
| SinButtonClicked | Boolean | This variable holds the Boolean for the Sin button. This lets the btnEquals\_Click method know which calculation to do. |
| CosButtonClicked | Boolean | This variable holds the Boolean for the Cos button. This lets the btnEquals\_Click method know which calculation to do. |

# Algorithms

## Number Algorithms

The number pad has very similar algorithms for each of the numbers in the calculator. One algorithm is added below for ease.

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

{

txtDisplay.Text = txtDisplay.Text + btnZero.Text;

}

## Clear Button

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

{

txtDisplay.Clear();

firstValue = 0;

secondValue = 0;

}

## Dot button

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

{

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

txtDisplay.Text = txtDisplay.Text + btnDot.Text;

}

## Set FirstValue

setFirstValue Algorithm was created to shorten the other methods when they needed to assign the first value to the firstValue variable. This algorithm also allows for the value from the text display to be parsed within a try-catch to stop errors from incorrect entries.

private void setfirstValue()

{

try

{

firstValue += double.Parse(txtDisplay.Text);//txtbox -> firstValue

txtDisplay.Clear();

}

catch (Exception x)

{

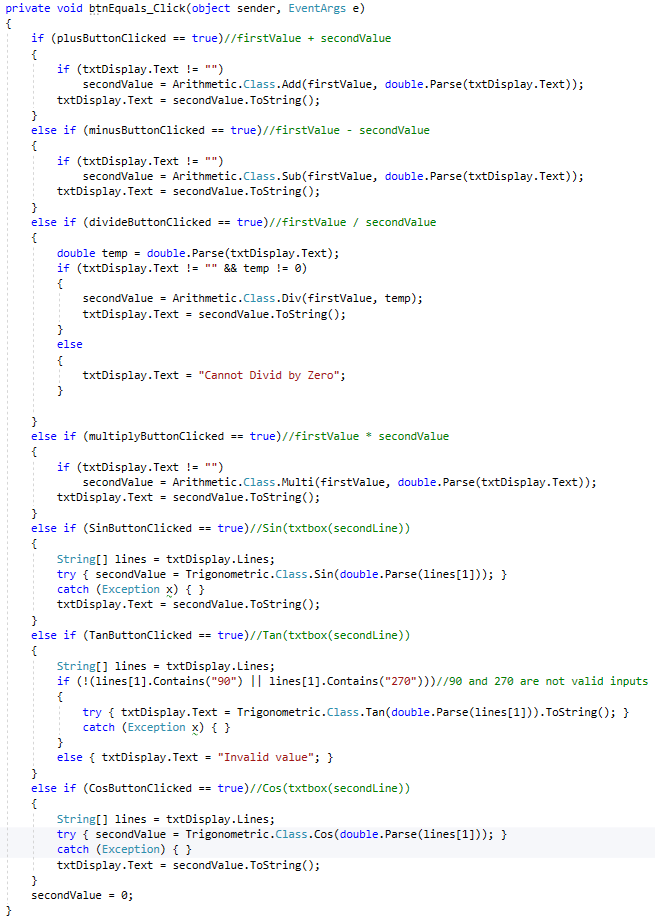
txtDisplay.Text = "Invalid Number" + x;//if parse throws error

}

}

## btnEquals

The btnEquals method goes through each ButtonClicked variable and runs the algorithm that is true. Only one variable will be true, and the else-if statements will find that one true variable and run its calculation to find the wanted output. At the end the secondValue variable is set back to zero.



## Arithmetic btn

The arithmetic button methods are very similar to each other where they use the setfirstValue method then clears the text box. The method then sets all buttonClicked variables to false except for the corresponding variable to the button. In this case it is the pluseButtonClicked variable. The other arithmetic methods are exactly the same other than which variable they switch to true.

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

{

setfirstValue();//set firstValue method

txtDisplay.Clear();

plusButtonClicked = true;

minusButtonClicked = false;

divideButtonClicked = false;

multiplyButtonClicked = false;

}

## Trigonometric btn

For the Trigonometric buttons they are again very similar to each other where they first clear the text display then sets all the buttonClicked variables to false other than the value corresponding to the method (in the case below SinButtonClicked). Finally, it prints the ‘Sin(‘ prompt on the first line then moves the curser to a new line. This works as a text prompt for the user and when the equals button is pressed the sin path will read the number from the 2nd line and use that instead of having to search a single line for the number. This is the same for the cos and tan paths too.

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

{

txtDisplay.Clear();

plusButtonClicked = false;

minusButtonClicked = false;

divideButtonClicked = false;

multiplyButtonClicked = false;

SinButtonClicked = true;

TanButtonClicked = false;

CosButtonClicked = false;

txtDisplay.Text = "Sin(";//Sin( prompt

txtDisplay.AppendText(Environment.NewLine);//next line

}

## Algebraic btn

Unlike the trigonometric and the arithmetic buttons, the algebraic buttons each have their own unique method and don’t use the buttonClicked values. They also are not included in the equals button which makes it so the result of the first click can be calculated again if the button is pressed twice.

### Cube Root btn

Once the cube root button has been clicked the String from the text display is parsed to a double value called ‘num’. This is wrapped in a try-catch to catch an error from the parse if something other than a number is entered in the calculator. Then if the number is above zero the double is put through the cube method in the Algebratic library. The result is then turned to a string and sent back to the text display.

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

{

double num = 0;

try//if no number is entered parse will cause error

{

num = double.Parse(txtDisplay.Text);

}

catch (Exception x)

{

txtDisplay.Text = "Number must be positive";

}

if (num >= 0)

{

//cube number from txtbox and return to txtbox

txtDisplay.Text = Algebratic.Class.Cube(num).ToString();

}

}

### Square Root btn

The square root button method parses the text display the same way as the cube root method. the difference between the two is that the number is run through the SquareRoot() method. More about this method can be found below but it will return the square root of a number, which in this case is turned back to a string and set back into the textbox. If the double is less than zero an error message box will open stating that the number must be above zero.

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

{

double num = 0;

if (txtDisplay.Text != "")

{

try//if no number is entered parse will cause error

{

num = double.Parse(txtDisplay.Text);

}

catch (Exception x)

{

txtDisplay.Text = "Number must be positive";

}

}

if (num > 0)

{

txtDisplay.Text = SquareRoot(num).ToString();

}

else

{

//To square root a number the number must be above zero

MessageBox.Show("Number must be above zero", "Error Message");

}

}

### Inverse btn

The inverse button method is much like the other Algebraic buttons where the text display is parsed to a double within a try-catch. After the number is checked to see if it is equal or larger than zero the double is put through the Inver method in the Algebratic library. This is wrapped in a try-catch because the inver method can throw an exception when the double entered is zero. This error is caught by a try-catch which surrounds the library method. if the exception is thrown the exception is converted to a string and printed in the text display. An try-catch was used to show that the library method can throw an error. If the code was to be simplified the if statement surrounding it could be changed to greater than zero, this way a zero or lower could not be entered. An ‘else’ could also be used for an custom error message to be printed to the text display.

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

{

double num = 0;

try//if no number is entered parse will cause error

{

num = double.Parse(txtDisplay.Text);

}

catch (Exception x)

{

txtDisplay.Text = "Number must be positive";

}

if (num >= 0)

{

//inver method can throw exception this is why a try catch is needed.

//exception is printed to txtdisplay if an error has occured

try { txtDisplay.Text = Algebratic.Class.Inver(num).ToString(); }

catch (Exception x) { txtDisplay.Text = x.ToString(); }

}

}

## Square Root btn

The SquareRoot method sends a double to the ‘SQRT’ method in the algebratic library. It then returns the result back.

private double SquareRoot(double x)

{

secondValue = Algebratic.Class.SQRT(x);

return secondValue;

}

# Testing Procedure

It is recommended that the calculator application is tested in two ways, with singular calculations(how its meant to be used) and with random button pressing/multiple inputs. For the singular calculations each equation will be tested with different inputs (for example: random number, decimal, zero). The singular test will also include test where a calculation is selected then another as well. This is to make sure they don’t create errors between them. For the multi test several buttons will be pressed at random to see if an error occurs. Between these two sets of tests it should cover every possible error that could occur.

# Future Upgrades

The Calculator application is not perfect and could be upgraded to perform more complex tasks. The fact that it can not do negative numbers is a major downfall for the calculator. This should be the first upgrade that is done to fully utilize the possibilities of the calculator. There are more suggestions for upgrades below.

* Multiple calculations in one equation

This suggestion refers to the fact that the application can only do one calculation in an equation at a time. For example, one plus one can be done but one plus one divide six is not possible.

* Negative numbers
* More Scientific calculations
* Additional types of calculators (Programmer, Date Calculation)