Programmer Documentation

Programmer Documentation

Contents

Programmer Documentation

Part 1. Requirements Documentation

Section 1. Introduction

1.1 --- Scope statement

The software being created by team 7 is a calculator with scientific functionality, which is stand alone and uses a GUI.

- Justification and Product Scope
  We are going to create a scientific calculator that is standalone for use in a professional environment. The justification would be to have a calculator that can implement more than the standard scientific calculator in a standalone executable format.
- Acceptance criteria and Deliverables
  The project will be completed by December 12th, which will be limited to a scientific calculator that is comparable to the TI scientific calculator. The project will come complete with a download link and instructional information on how to use the calculator along with conceptual graphs and charts to show data flow.
- Product exclusion and Constraints
  The calculator is being thought of in terms of the GUI viability and ability to handle order of operations. The scope of the calculator is limited to basic functionality with some added features.
- Assumptions
  The main project constraint is time as the team only has a limited amount of time to code and produce the product. Updates and revisions will not be implemented after the final product is presented.

1.2 --- Definitions, acronyms, and abbreviations

This document describes all the documentations and some of the charts to explain the programming process that team 7 has for the whole project. Including the general description of the project, design documentations (the diagrams, pseudocode, tables), source code documentation, testing documentation, and dealing with bugs and issues.

1.3 --- References

References the StringToExpression Class

https://stackoverflow.com/questions/21750824/how-to-convert-a-string-to-a-mathematical-expression-programmatically

Section 2. General Description

Programmer Documentation

2.1 Product perspective

This software serves a small platform to build a wider calculation application on top of it. The initial version will only have small features such as

1. Standard Arithmetic Operations (+/-*)
2. Parsing strings for expressions
3. Friendly interface
4. History function to track user input

so that over time more components can be added, to make a large production level app. The program at this level only offers use to a test audience and other developers considering that it offers less functionality than a standard windows calculator (which is not saying much given how many features it has). Over time more features will be added such as

1. Graphing
2. A more robust history window (tracks past results like windows calculator)
3. Linear equation solver.

2.2 Product functions

The current build of the calculator serves to compute complex arithmetic expressions as it’s main functionality. Basic arithmetic operations, some trigonometric, and

2.3 User characteristics

The current end users of this software are developers. The components of this calculator do not offer more functionality, in its current state and possibly future state, than the standard windows calculator. Being that this calculator can only be run on windows hardware at this point in time, it is hard for anyone to imagine that outside of tech savvy users, and software developers that anyone would use this product over windows standard calculator. Especially given the wide array of functionality that comes with the standard calculator.

2.4 General constraints

The calculator can only run on a windows framework. The programmer must have Visual Studio, and the target audience must have .NET runtime installed in order to run the program. The current design of the program contains some inflexible features that in future versions should be modified (in this case the logic behind the calculator button binding is tied towards the button's variable name).

2.5 Assumption and dependencies

Supported Operating Systems:

- Windows 10
- Windows 8, 8.1
- Windows 7
- Windows Vista
- Windows Server 2008
- Windows Server 2008 R2
- Windows XP SP2, SP3
- Windows Server 2003 (excluding IA-64)

Development Environment:

- Microsoft Visual Studio 2010/2012/2013/2015/2017/2019
- .NET Framework. The following versions are supported:
    - .NET 4.0
    - .NET 4.5
    - .NET 4.5.1
    - .NET 4.5.2
    - .NET 4.6
    - .NET 4.7
    - .NET 4.8
    - .NET Core 3.1. Supported since R1 2020. Versions between R1 and R3 2019 support .NET Core 3.0.
    - .NET 5.0 Preview. Supported since R2 2020.

Hardware Environment:

- Processor: x86 or x64 1 GHz Pentium processor or equivalent (minimum); 1 GHz Pentium processor or equivalent (recommended)
- RAM: 512 MB (minimum); 1 GB (recommended)
- Hard disk: up to 1.5 GB of available space may be required
- Display: 800 x 600, 256 colors (minimum); 1024 x 768 high color, 32-bit (recommended)

This program was built using visual studio’s .NET Framework. To run and modify code it is assumed that both the user and the programer have the .Net runtime installed. The developer needs to additional have the visual studio suite installed to be able to run the code. There aren’t any extra dependencies needed to run the code outside of the standard visual studio suite. To be able to modify the code correctly the developer needs to be aware of how WPF applications are written. WPF applications are written in:

1. XAML
2. C#

where XAML handles the designs and C# handles the logic, with few exceptions. The entire interface of the calculator is written in XAML and the logic behind the calculations are written in C#. So a baseline knowledge of wpf applications are needed in order to understand this code.

Section 3. Specific Requirements

1. The user should have an interface to input there arithmetic expressions

    1.1. The user should be able to input digits and decimals into the number container.

    1.2 The user should be able to input standard arithmetic operations (+,-,/,*,^).

    1.3 The user should be able to input extended arithmetic operations (sin,cos,ln,log,%, etc)

    1.4 The user should be able to clear the calculator

    1.5 The user should be able to delete the digits they have input

    1.6 The user should be able to input standard mathematical constants (e, pi)

    1.7 The user should be able to resolve their arithmetic expression.

    1.8 The user should be able to input parenthesis to express a more nuanced arithmetic expression.

2. The user should be able to see the results of their inputs in the number container and history container.

    2.1. When inputting any digit the number container should be updated to reflect the input of that digit. (Input's of new digits should supersede other operating states)

    2.2 When inputting a binary operation (operations that take two inputs) the number in the number container should be resolved to the first input and leave the second input dangling for future values.

    2.3 When inputting repeated monomial operations the operations should stack until they are interrupted by a non-monomial operation.

    2.4 When inputting the delete operation the number container should resolve the difference between the following scenarios:

        2.4.1. Number container, containing user generated digits. In which the delete operation simply removes digits outside of the following scenarios:

            a. The container only contains a single digit (positive/negative) number. In which the container returns to the default value 0.

b. The container contains a decimal number with only 1 number in front of the decimal. In which case the container deletes the number and the decimal.

2.4.2. Number Container - Containing a calculator generate number where examples of such generated contents include:

i. Monomial Operations
ii. Resultant Operation
iii. Mathematical Constants

2.5. When inputting mathematical constants the number container should be replaced with that constants value.

2.6. When inputting parentheses the users should not be able to make simple mistakes.

2.6.1 When inputting a right parentheses there should not be more right parentheses than left.

2.6.2 When inputting a left parentheses immediately after a right parentheses. The program should assume that the user is asking implying multiplication.

2.6.3 When inputting a right parentheses after a left parentheses the program should not leave a dangling binomial operator nor should allow empty expressions inside of the parentheses. In both cases the users input inside the number container should be used and defaulted.

2.7 When resolving the user's expression the calculator should not leave dangling operators or parentheses in the algebraic expression. (Prior to being resolved).

3. The windows should implement a modern design.

3.1 Number container

3.1.1 The number container should have a dark large text font that is very clear and easy to distinguish.

3.1.2 The number container should have white background

3.2 History container

3.2.1 The top panel should implement Implement a dark design

3.2.2 The number container should have white background

3.3 Top Panel

3.3.1 The top panel should implement Implement a dark design

3.3.2 The top panel contain a menu button that opens the side panel

3.3.2.1 The menu button should animate on click.

3.4 Side Panel

3.4.1 The side panel should contain extra options to be implemented in future releases.

3.4.2 The side panel should animate open when the menu button is clicked.

3.5 Button Grid (and Buttons)

3.5.1 The button grid should contain all the buttons associated with the arithmetic side of the calculator.

3.5.2 Make sure the font on the buttons is uniform and readable.

3.5.3 Make sure the button grid contains all button representations of all of the following expressions and operations:

a. 0-9
b. .
c. sin/cos
d. + / * - ^
e. modulus (%)
f. absolute value (abs)
g. negation (neg)
h. ln
i. Mathematical Constants (e and pi)
j. Clear
k. Delete
l. =

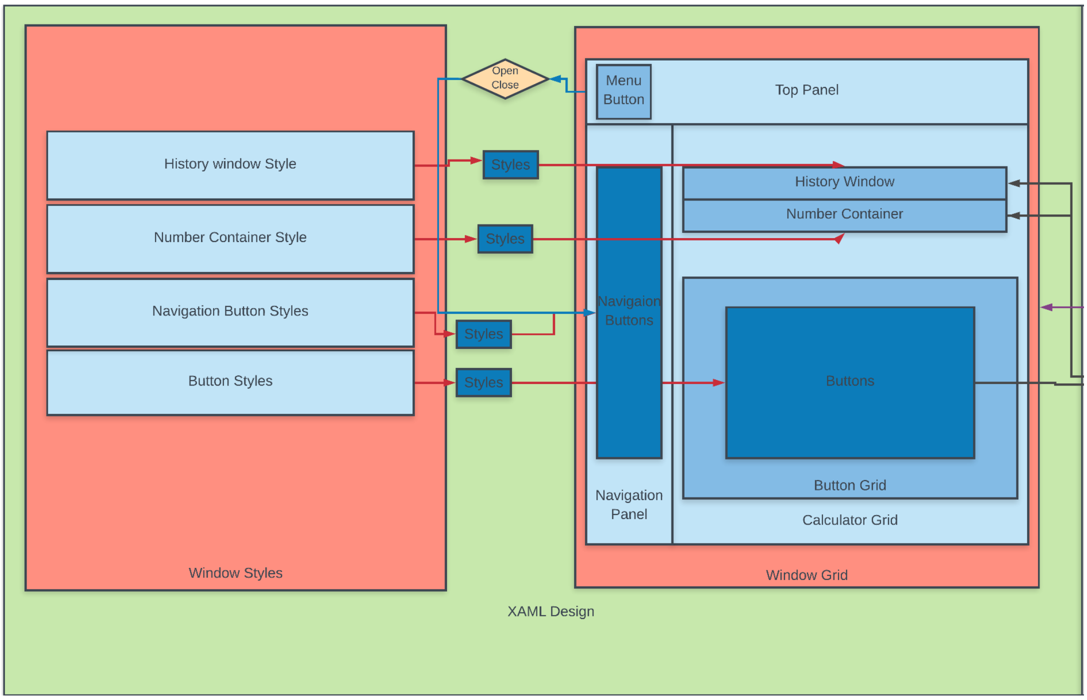
Programmer Documentation

Part 2. Design Documentation

Section 1. Architecture Diagrams

The program is broken into two big chunks. The C# code and the XAML code. The majority of the design is governed by the XAML diagram as shown below:

(Requirements: !._._/3._._ (All of them))



The main window is broken into broken into three parts which are the:

1. Top panel
2. Side panel
3. Calculator Grid

The top panel is there for more aesthetic reasons than functional. It holds the menu button which opens the side panel. The side panel holds the future options to be added to the program, here are some options that could be added in the future:

1. Settings
2. Graphing
3. Conversion
4. Memory

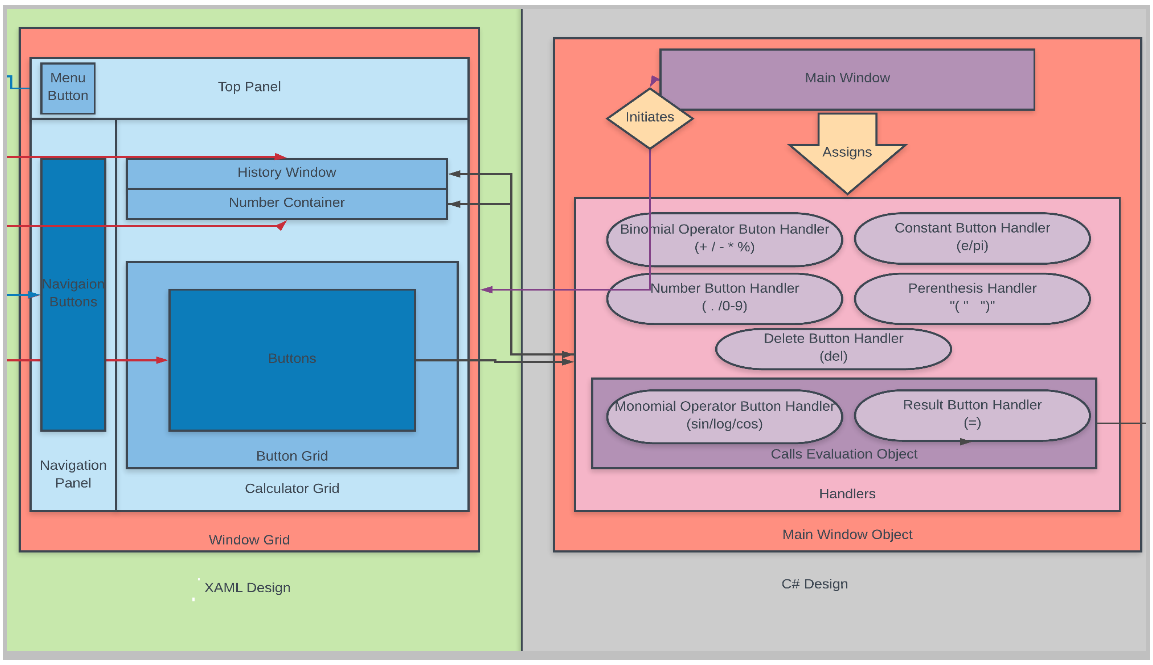
Similar to the windows calculator. The calculator grid holds the meat of our calculations and is divided into:

1. History Container
2. Number Container
3. Button Grid

The history container and number container show response to user inputs from the buttons found in the button grid. All State changes happening towards those two containers should involve the user interacting with the buttons in the button grid.

Section 1.2 C# and Xaml Integration Diagram

WPF applications have a nice synergy between c# code and html/css like XAML code. When the program is launched it initiates a Main Window Object from from the main window class. The main window object then binds all of the buttons on the button grid to their particular class of button handlers.


Menu Button
Top Panel
History Window
Number Container
Navigaion Buttons
Buttons
Button Grid
Navigation Panel
Calculator Grid
Window Grid
XAML Design
Main Window
Initiates
Assigns
Binomial Operator Buton Handler (+ / - * %)
Constant Button Handler (e/pi)
Number Button Handler (. /0-9)
Perenthesis Handler "(" ")"
Delete Button Handler (del)
Monomial Operator Button Handler (sin/log/cos)
Result Button Handler (=)
Calls Evaluation Object
Handlers
Main Window Object
C# Design

Programmer Documentation

Section 1.3 C# Design:

The C# design work by :

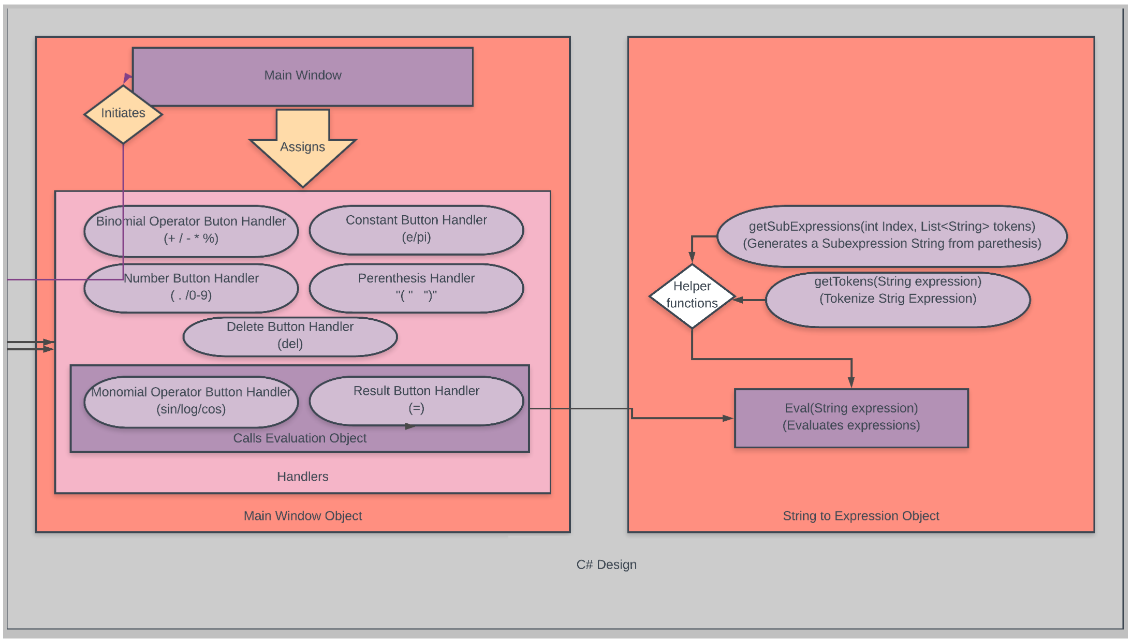
1. Setting the state values behind the scenes
2. Defining handlers for all the buttons presented in the Button Grid
3. Processing expressions in the history container and translating those into values presented in the numeric container.

All of the button handlers operate off of two classes of information:

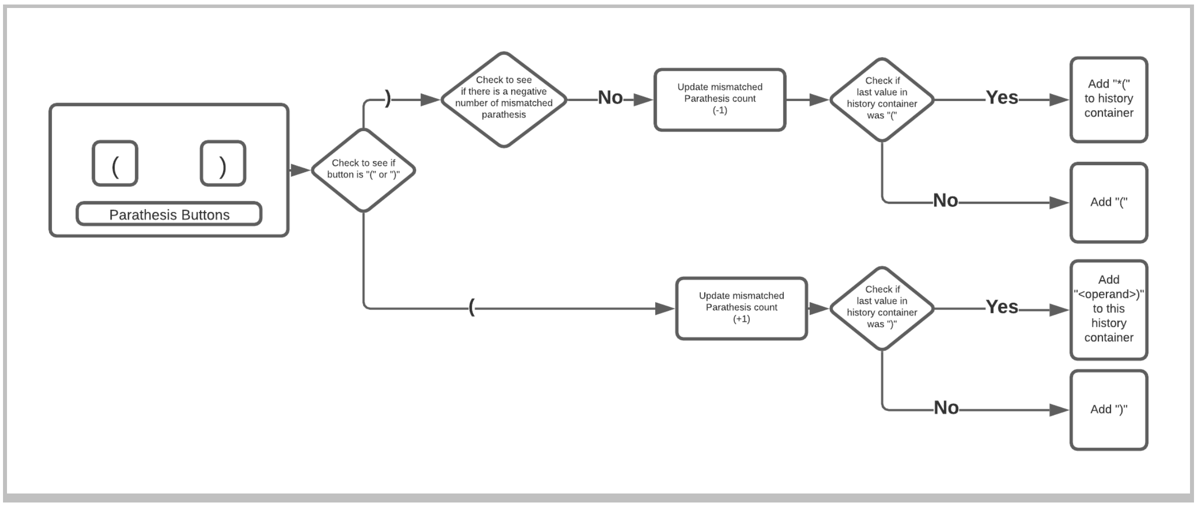
1. Global state variables.
2. Current values in the containers.

Depending on the values of those two certain actions either are or aren't possible. With in the C# code there is a class that specifically handles the user generated expressions. It deals with all of the messy evaluation of arithmetic expressions.

(Requirements: 1._._/ 2._._ (All of them))

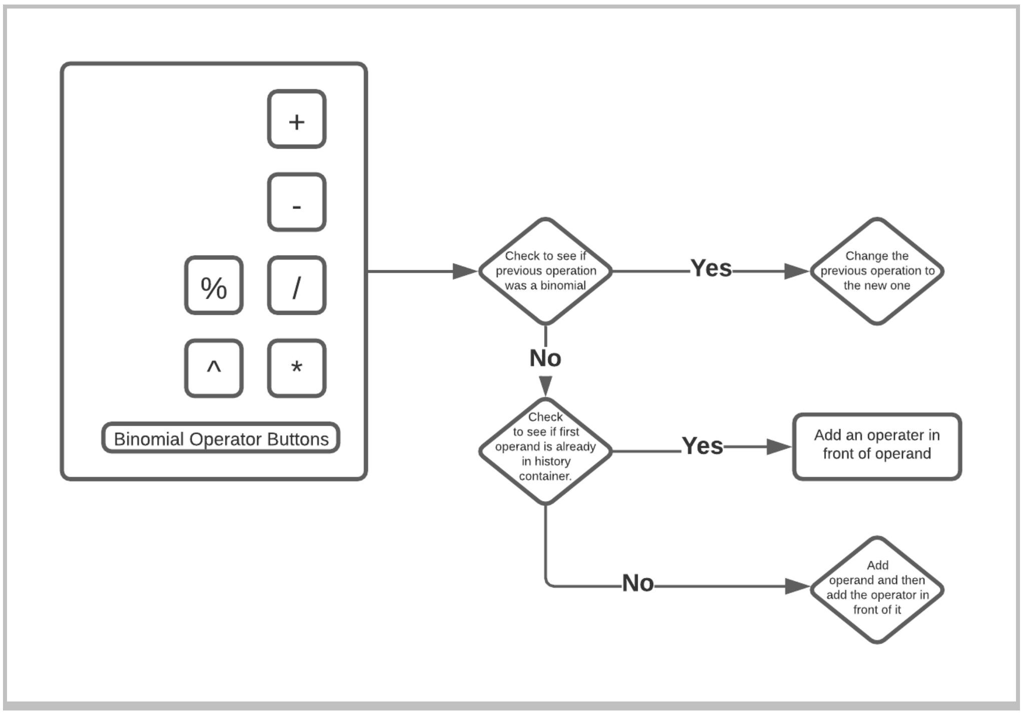


(Requirement: 1.8/2.6)

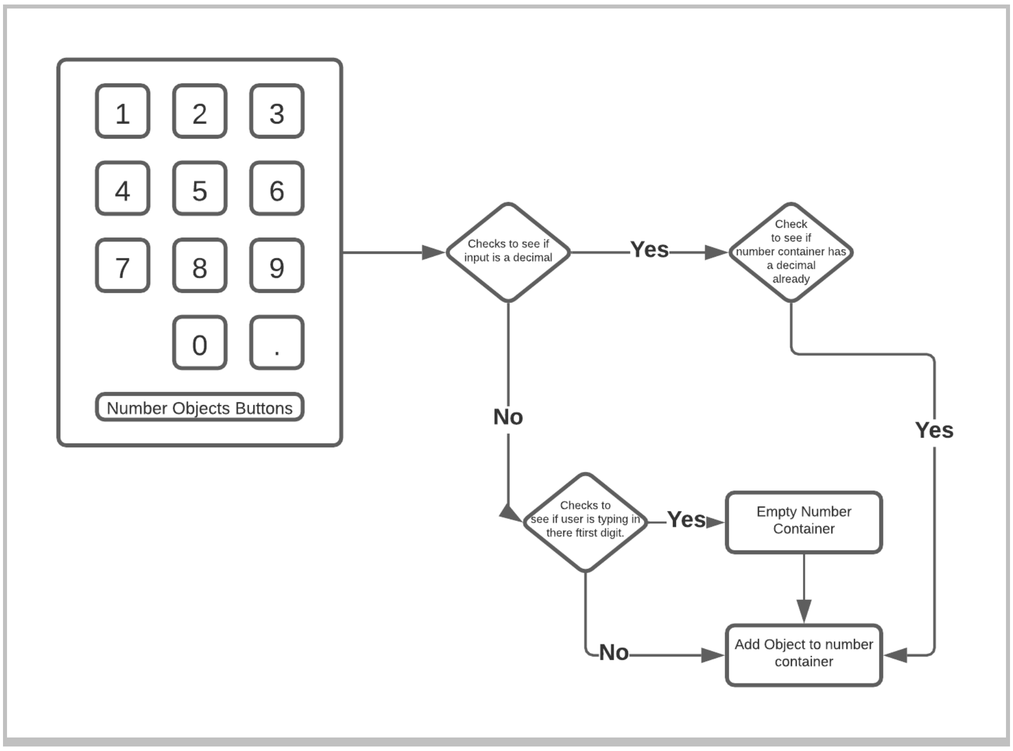

(
)
Parathesis Buttons
Check to see if button is "(" or ")"
)
Check to see if there is a negative number of mismatched parathesis
No
Update mismatched Parathesis count (-1)
Check if last value in history container was "("
Yes
Add "*(" to history container
No
Add "("
(
Update mismatched Parathesis count (+1)
Check if last value in history container was ")"
Yes
Add "<operand>)" to this history container
No
Add ")"

Programmer Documentation

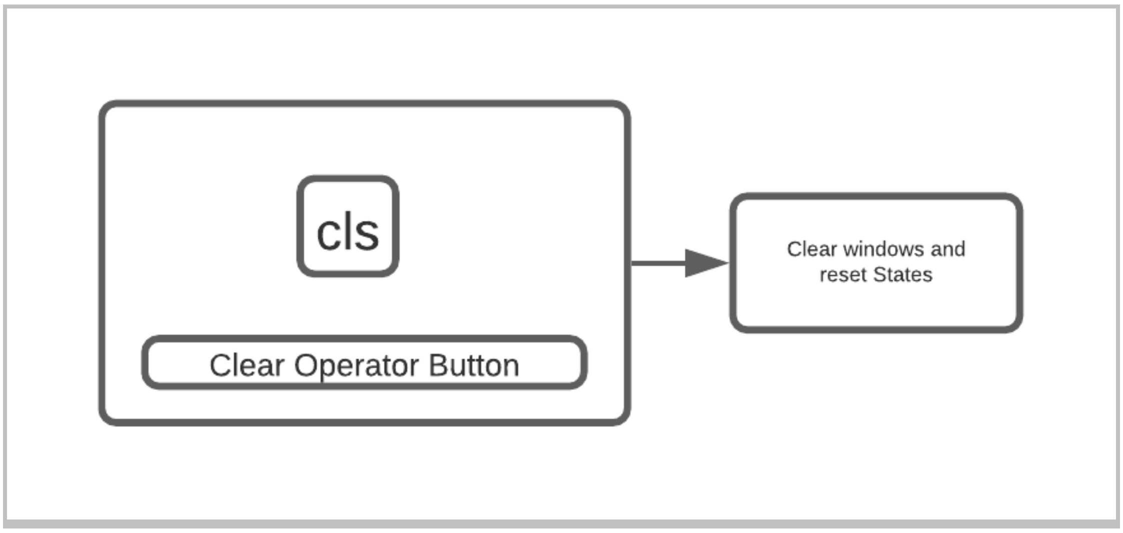
(Requirements: 2.2 and 1.3/1.4)


+
-
%
/
^
*
Binomial Operator Buttons
Check to see if previous operation was a binomial
Yes
Change the previous operation to the new one
No
Check to see if first operand is already in history container.
Yes
Add an operater in front of operand
No
Add operand and then add the operator in front of it

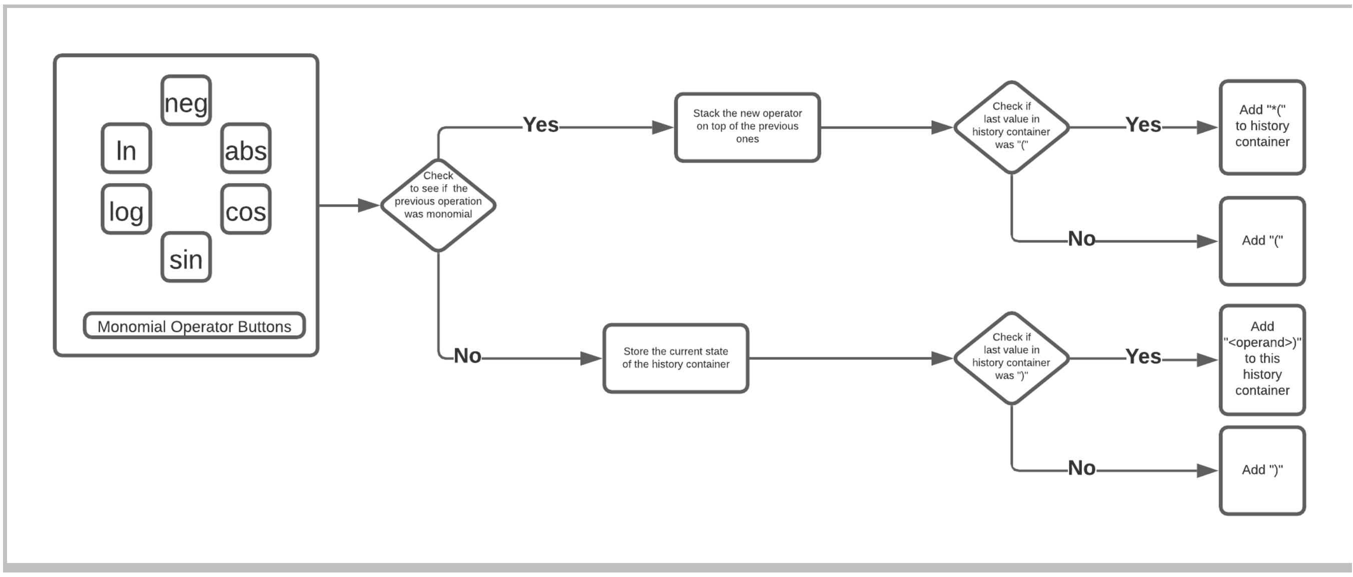
(Requirements 1.1)


1
2
3
4
5
6
7
8
9
0
.
Number Objects Buttons
Checks to see if input is a decimal
Yes
Check to see if number container has a decimal already
No
Yes
Checks to see if user is typing in there ftirst digit.
Yes
Empty Number Container
No
Add Object to number container

(Requirement: 1.4)

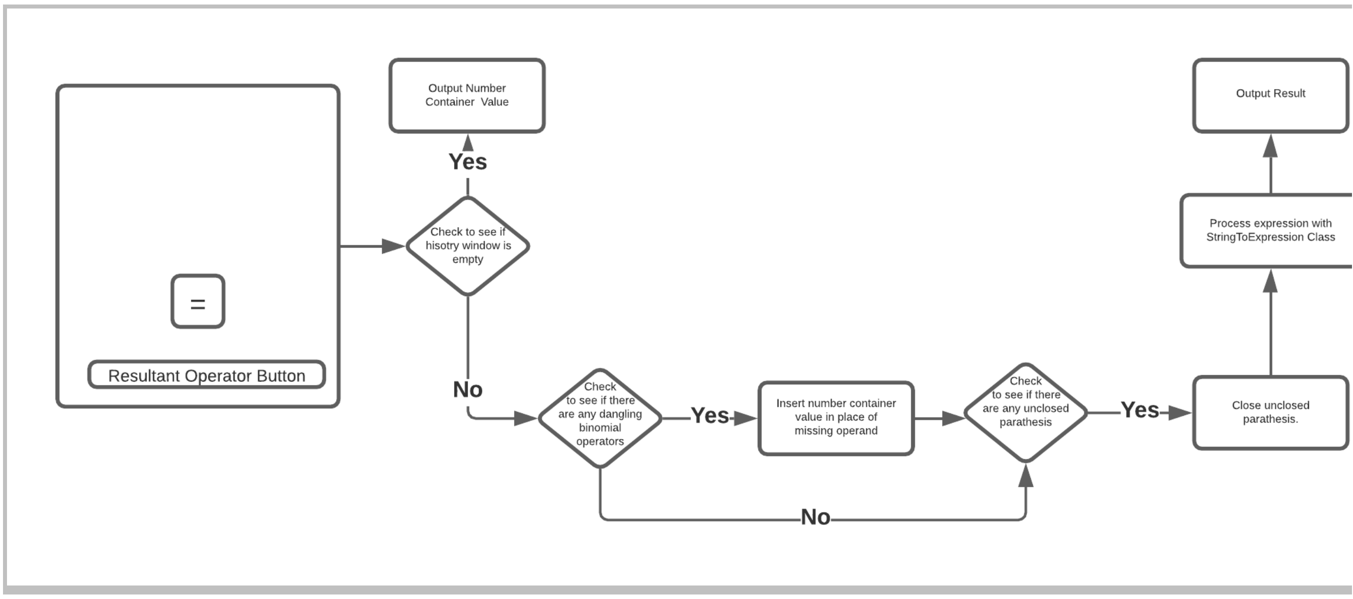


(Requirements: 1.3/1.5)


neg
ln
abs
log
cos
sin
Monomial Operator Buttons
Check to see if the previous operation was monomial
Yes
Stack the new operator on top of the previous ones
Check if last value in history container was "("
Yes
Add "*(" to history container
No
Add "("
No
Store the current state of the history container
Check if last value in history container was ")"
Yes
Add "<operand>)" to this history container
No
Add ")"

Programmer Documentation

(Requirements: 2.7/1.7)



=
Resultant Operator Button
Output Number Container Value
Yes
Check to see if hisotry window is empty
No
Check to see if there are any dangling binomial operators
Yes
Insert number container value in place of missing operand
No
Check to see if there are any unclosed parathesis
Yes
Close unclosed parathesis.
Process expression with StringToExpression Class
Output Result

Programmer Documentation

Part 3. Source Code Documentation

The source code as well as pseudocode are all within the uploading zip file. GitHub is where the source code repository can be found. Use the link below to access.

GitHub – aaronharrell12/Calculator2

Part 4. Testing Documentation

Test case #	Requirement Tested	Rationale	Input(s)	Expected Output	Pass/Fail
1	1.1	Users should be able to input digits.	User selects a button on numeric button or digit on the calculator	Number container updates to show number and digit	PASS
2	1.2	Users should be able to input binomial operators.	User selected Arithmetic operators	Operator should be displayed in a way that doesn't violate	PASS
3	1.1 and 1.2	User's values that generated in the number container should be displayed in history container	User inputs "3" and then inputs basic operator "+"	History container shows value of "3+"	PASS
4	1.3	Users should be able to input extended arithmetic expressions.	User select "2" and then clicks "sin"	History containers shows "0.909297"	PASS
5	1.4	User should be able to clear the calculator	User selects clear.	The Number container and History Container should be cleared.	PASS

6	1.5	Users should be able to delete the digits they have input	User enters “41”and presses delete	Number container shows “4”	PASS
7	2.2	The history container should display values in a consistent way.	The user select “3” and then select “+”	The Operator should show the value of the mathematical constants	PASS
10	2.3	The users repeated monomial operations should stack.	The users select “3” and select “sin” and “cos”	The history window should show “cos(sin(3))”.	PASS
9	2.4	Users should be able to input parenthesis to express a more nuanced arithmetic expression.	User selects parentheses along with the digits and hit “=”	The Operator should show the same value they entered	PASS
10	2.1	When inputting any digit, the number container should be updated to reflect the input of that digit.			PASS

11	2.2	When inputting a binary operation (operations that take two inputs) the number in the number container should be resolved to the first input and leave the second input dangling for future values.			PASS

Part 5. Bugs and Issues

1. If the user type in “1/0” infinity comes out in the resultant. Certain calculator functions break under this input. Example: “infinity - 3”.
2. Inserting a function before a value the program computes the wrong result.
3. Inserting(neg) before a value the program will output 0.