# Design Changes

**Original Version**

## 2.3 Internal Data Structure

*The Text Interface module will utilize a list to store the command line arguments entered by the user.*
*argList[i]*
*Each element in the list is a string containing the name of the program followed by arguments which include the name of the input file.The value of the index ranges from 0 to 1.*

*Strings in the argList[] list which are relevant to the program are:*

| ***Index value*** | ***Contents*** |
|---|---|
| *0* | *contains the name of the program being run* |
| *1* | *contains the name of the input file to be read by the program* |
| *2 or >2* | *not used. If length of the command line arguments is greater than or equal to 2, the program will exit, notifying the user that their arguments exceed the required length.* |

*Format of the string stored in argList[0] and argList[1]*
*<name of the program> /<filepath>/<filename/>*

*<name of the program>* - *the source file which contains the code to run the program. For instance, "./test.py". "./' indicates that the program is being run from the current directory, followed by "test" which is the name of source file and lastly the extension "py" which indicates that is a python source file.*

*<filePath>* - *a forward slash delimited string containing the path name which leads to the location of the file <filename>.*

*<filename>* - *name of the file entered by the user normally is a .txt file.*

**Revised Version**

The list will be parsed further, using the extractFileName function in order to extract the name of the file from the file path. The file name will then be stored into a string variable called fileName, which will be used to search for the file, open the file and read from the file.

The internal data structure was changed to:

There are no command line arguments throughout the entire project; everything is done through a menu.

The text interface menu utilizes a display menu in the terminal, which gives the option:
1) press 'f' for file
2) press 'm' for manually entering the mathematical expression
3) quit

The first option takes in a file which prompts the user and looks to see if that file exists or not, if the file exists it will evaluate the data in the file and print the result to the screen. If the file does not exist or the file could not be calculated properly it will print out some error statement or "0" based on the input.

The second option will allow the user to manually enter the mathematical expression that will be evaluated if possible and printed to the screen, otherwise it will print out some error statement or "0" based on the input.

**Original Version**

*The Text Interface module provides the user with an option of using the graphing calculator on the terminal rather than the GUI. This module consists of all the functions that will be used if the user wishes to run the program on the text interface. They include functions for reading the input from a text file, for getting the input manually, extracting the filename and filepath from the file path, and lastly a menu function which will ask the user whether they want to enter the input manually or via text file.*

### *3.0.1 Interface Description*

*textInterfaceOptions()*

- *Does not return anything.*
- *Does not take in any parameters.*

*Algorithmic Model*

1. *Display options for (1) Manual Input or (2) File Input to the terminal.*
2. *Get input from the user.*
3. *Store user input into a integer variable called 'choice'*
4. *Error check to make sure 'choice' is equal to '1' or '2', if not equal to '1' or '2' prompt user for input again until correct input is entered, otherwise continue.*
5. *If choice is equal to 1 then branch of to the getManualInput() function, otherwise continue.*
6. *If choice is equal to 2, then branch of to the getFilePath() function.*

*Local Data Structures*

- *None*

*Restrictions and Limitations*

- *None*

*Performance Issues*

- *None*

*Design Constraints*

- *None*

**Revised Version**

### 3.0.1 Interface Description

- Changed name of textInterfaceOptions() to textInterfaceMenu().
- Switched the order of option 1 and option 2.

Algorithmic Model

1. Displays options for (1) File Input to the terminal or (2) Manual Input.
2. *Get input from the user.*
3. *Store user input into a integer variable called 'choice'*
4. Error check to make sure 'choice' is equal to '1' or '2', if not equal to '1' or '2' prompt user for input again until correct input is entered, otherwise continue.
5. If the user will choose option 1, it will call function readFile(input) which takes the filename and checks if the file exists or not. If it exists, the function will try to calculate the expression in the readFile function and print the result.
6. If the choice is equal to 2, it will prompt for a string and evaluate it right on the spot and give out a proper result.

Local Data Structures

- None

Restrictions and Limitations

- None

Performance Issues

- None

Design Constraints

- None

Function names were changed because the new names appeared to be more correct in the naming format for our project.

**Original Version**

### 3.0.2 Interface Description

*mathEquation getManualInput()*

- *The function returns a string variable called mathEquation, which contains the mathematical equation to be evaluated by the calculator.*
  *The function does not take in any parameters*

*Algorithmic Model*

1. *Prompt the user to enter a mathematical equation manually*
2. *Store the expression in a string variable called 'mathEquation'*
3. *Call errorCheck( ) function using the variable 'mathEquation' as a parameter to check if the mathematical equation stored in variable expression is valid.*
4. *If the expression is valid, then return mathEquation.*

*Local Data Structures*

- *None*

*Restrictions and Limitations*

- *None*

*Performance Issues*

- *None*

*Design Constraints*

- *None*

**Revised Version**

All of the calculations were done right away in Section 3.0.1 while the user is choosing option 2.

- *getManualInput* was removed from the project.

**Original Version**

### *3.0.3 Interface Description*

*mathEquation  fileRead(fileName, filePath)*

- *Returns a string variable called mathEquation, which contains the mathematical expression, read from the file.*
- *Takes in a string variable called fileName which contains the name of the input file extracted from the getPathandName( filePath) function.*
- *Takes in a string variable called filePath which contains the file path extracted from the getPathandName(filePath) function.*

***Algorithmic Model***

1. *Set counter variable to 0 to count the number of lines in the input file.*
2. *If the fileName is equal to null, then return an error message notifying the user that the file is empty and exit to textInterfaceOptions( ) function.*
3. *If the fileName is not a valid file or if it does not exist then return an error message to notify the user that the file does not exist and return to the textInterfaceOptions() function otherwise continue.*
4. *If the filePath is not a valid path or if it does not exist then return an error message to notify the user that the file path does not exist and return to the textInterfaceOptions() function otherwise continue.*
5. *If both the fileName and filePath are valid and they exist, then locate the file by using the filePath variable.*
6. *Open the file by using the fileName variable, which contains the name of the file.*
7. *Read the input from the file and store the input in the string variable mathEquation.*
8. *Add one to the counter.*
9. *If counter is greater than one, then return an error message to notify the user that file contains more than one line and return to the textInterfaceOptions() function otherwise continue.*
10. *Call the errorCheck(mathEquation) function  using the fileInput as a parameter to check if the fileInput is a valid mathematical expression.*
11. *If it is valid, then return mathEquation.*

***Local Data Structures***

- *None*

***Restrictions and Limitations:***

- *The function cannot read more than one line, the program will return to the textInterFaceOptions().*

***Performance Issues:***

- *None*

***Design Constraints***

- *None*

**Revised Version**

### 3.0.3 Interface Description

The error checking stayed the same, except any error checking was done inside the fileRead function and it does not call any other functions.

The readFile() function only evaluates the last string in the file and prints the result, it does not use a list or any other temporary variables.

**Original Version**

### *3.0.4 Interface Description*

*getPathandName(filePath)*

- *Receives a list filePath which contains the path of the file along with the name of the file. For instance, ['test.py', '/home/example_user/example_directory/test.txt']*
- *Does not return anything.*

*Algorithmic Model*

1. *Extract the file path from the list[1] and store it into a string variable called fPath.*
2. *Extract the file name from list[1] and store it into a string variable called fName.*

*Restrictions and Limitations*

- *None*

*Performance issues*

- *None*

*Local Data Structures*

- *Uses a list, which stores the command arguments.*

*Design Constraints*

- *None*

**Revised Version**

### *3.0.4 Interface Description*

This function was removed from the project and was not used. The readFile(input) function checks if the file exists and if it is accessible through the input parameter.

**Original Version**

### 3.0.5 Interface Description

*getCommandArgs()*

- *Does not return anything.*
- *Does not receive anything.*

*Algorithmic Model*

1. *Prompt the user to enter a file path*
2. *Store the user input into list variable called argList.*

*Restrictions and Limitations*

- *None*

*Performance issues*

- *None*

*Local Data Structures*

- *Uses a list, which stores the command arguments.*

*Design Constraints*

- *None*

**Revised Version**

### 3.0.5 Interface Description

This function was not used in the project, so it was removed. Instead, the algorithmic model below was done on the menu.

Algorithmic Model

1. Prompt the user to enter a file path
2. Allows user to manually input a mathematical expression.

Command line arguments were not used throughout this project.

**Original Version**

### 3.1.1 Interface Description

*The common functions module manages all the functions that are shared and are common amongst the Graphical User Interface and the Text Interface. It includes functions like errorCheck(inputString), calculations(inputString), graphFunction(inputString) and lastly, printHistory().*

*valid errorCheck(inputString)*

- *This function receives a string variable called inputString, which contains a mathematical expression.*
- *Returns a valid value of true if inputString is valid and false if it is not.*

*Algorithmic Model*

1. *Set 'operations1' to the tuple containing (['+', '-', '*', '/', '(', ')'])*
2. *Set 'operations2' to the tuple containing (['sin', 'cos', 'tan', 'pi', 'ln', 'log', 'e', 'sqrt'])*
3. *Set length to length of inputString, which contains the mathematical expression entered by the user either through manual input or through a text file.*
4. *Set index to zero*
5. *For all the characters in the inputString*
   *5.1 If the character at index of inputString does not belong to the set "operations1" or if the characters at index of inputString does not belong to the set "operations2" then display a an error message to the user notifying them that the inputString is invalid.*
   *5.2 set the boolean variable valid to false and return valid.*
6. *If the inputString is valid, set the boolean variable valid to true and return valid.*

*Restrictions and Limitations*

*This function will not be able to protect against something that belongs to the set, but is an incorrect mathematical expression for instance, "sin+-Cos".*

*Performance Issues*

*The function may run slow depending on the size of the file and the length of the mathematical expression. It might crash if it encounters invalid input such as the one states in restrictions and limitations.*

*Local Data Structures:*

*This particular function utilizes two tuples to store basic mathematical operation characters "+", "-", "/" and "*", and mathematical key words sin, cos, tan, ln, log,e,pi etc. They will be used to check for the validity of the mathematical expression. Through each iteration, a character from the input string will be compared to each of the tuples and if the mathematical operation does not belong to any of the two tuples then an error message will be displayed to the user letting them know about the incorrect expression.*

**Revised Version**

### 3.1.1 Interface Description

errorCheck() function was removed from the project and not used. This errorCheck was done inside the evaluation in the calculations(input) function.

**Original Version**

### 3.1.2 Interface Description

*evaluatedString  calculations(inputString)*

- *The functions returns a string variable called 'evaluatedString' which contains the information needed to graph the function*
- *The function receives a string variable 'inputString' which contains the mathematical expression to be evaluated*

*Algorithmic Model*

1. *Evaluate inputString using order of operations and the built -in python eval function.*
2. *Store value in string 'evaluatedString'*
3. *Call storeHistory function with inputString and evaluatedString as parameters*
4. *Return 'evaluatedString'*

*Restrictions and Limitations*

- *Cannot evaluate functions with a large range*
- *Cannot evaluate in terms of scientific notation*

*Local Data Structures*

- *None*

*Performance Issues*

- *If the expression is too complex it will take too long to evaluate and cause the system to be slow*

*Design Constraints*

- *None*

**Revised Version**

### 3.1.2 Interface Description

The function name calculations(input) was changed to calculateResult(input) as the new naming method is more clear and meaningful. This function takes in the total string from the display box and evaluates the result using eval() from python library. All the error checking is done in this function and gives out result based on the input. Any error will print an error message or output "0" to the display box.

**Original Version**

### 3.1.3 Interface Description

*graphFunction(mathExpression)*

- *This function does not have a return value.*
- *This function receives a string variable call 'mathExpression' that contains the expression that is to be graphed.*

*Algorithmic Model*

1. *Set the result variable to an empty string*
2. *Set the variable xRange to zero*
3. *Call the errorCheck function with 'mathExpression' as the parameter, to check if the mathematical equation is correct.*
4. *Define a range of 0 -9 for the range using the np.linspace() function to initialize size of graph. This function will evaluate all the possible values between the range of 0 -9 and will plot the graph based on the paramters of the np.linspace() function.*
5. *set mathExpression equal to the yRange variable.*
6. *Use the mathExpression as the yRange and the xRange variable as parameters to the plt.plot() function.*
7. *display the graph using the plt.show() function.*

*Restrictions and Limitations*

- *None*

*Local Data Structure*

- *None*

*Performance Issues*

- *Size of range will affect speed of program when 'drawing' the graph.*

*Design Constraints*

- *None*

**Revised Version**

### 3.1.3 Interface Description

graphResult(mathExpression)

Algorithmic Model

1. Set the result variable to an empty string
2. Set the variable xRange to zero
3. Call the errorCheck function with 'mathExpression' as the parameter, to check if the mathematical equation is correct.
4. Define a range of 0 -9 for the range using the np.linspace() function to initialize size of graph. This function will evaluate all the possible values between the range of 0 -9 and will plot the graph based on the paramters of the np.linspace() function.
5. set mathExpression equal to the yRange variable.
6. Use the mathExpression as the yRange and the xRange variable as parameters to the plt.plot() function.
7. display the graph using the plt.show() function.

Restrictions and Limitations

- None

Local Data Structure

- None

Performance Issues

- Size of range will affect speed of program when 'drawing' the graph.

Design Constraints

- None

Plotly graphing library was used in the graphing calculator.

**Original Version**

### *3.1.4 Interface Description*

*storeHistory(expression, result)*

- *This function does not return any values*
- *This function receives the a string called 'expression' which contains an evaluated expression (ex; 5+3=8).*

*Algorithmic Model*

1. *Open 'history.txt' file*
2. *Append 'expression + "=" + result' to the end of the file*
3. *Close the file*

*Local Data Structures*

- *None*

*Restrictions and Limitations*

- *None*

*Performance Issues*

- *None*

*Design Constraints*

- *None*

**Revised Version**

### 3.1.4 Interface Description

There is only one history displayed at a time in the display box, which is only the current history. The storeHistory(expression, result) function was not used, so it was removed.

**Original Version**

### *3.1.5 Interface Description*

*printHistory()*

- *This function does not return any values*
- *This does not have any parameters.*

***Algorithmic Model***

1. *Open 'history.txt' file*
2. *Read file*
3. *Print file*

***Local Data Structures***

- *None*

***Restrictions and Limitations***

- *None*

***Performance Issues***

- *None*

***Design Constraints***

- *If file is corrupt the history will be lost*

**Revised Version**

### 3.1.5 Interface Description

printHistory() function was removed from the project as it was not needed. The display box of the GUI constantly shows all the information needed for the user.

**Original Version**

### *3.2 Description of GUI function*

- *Provides the an interactive interface for the user which resembles a real-life graphing calculator. Only contains one function, which is displayGui()*

### *3.2.1 Interface Description*

*displayGUI( )*

- *Does not take any paramters and nor does it return anything.*

*Algorithmic model*

1. *Display the GUI for the use to see.*
2. *Inputs from the GUI are calculated by storing the input in a string variable called inputString and using that as a parameter for the calculations(inputString) function.*
3. *Sends the inputString variable to the graphingFunction(inputString) function using the inputString as the parameter.*

*Local Data Structures*

- *None*

*Restrictions and Limitations*

- *None*

*Performance Issues*

- *None*

*Design Constraints*

- *None*

**Revised Version**

The GUI is imported at the start of the program. All of the buttons are initialized at the start of the function and the buttons link themselves to specific functions.

Example:

Button "=" links to calculateResult() function.

There is no displayGUI() function, and it was removed from this project.

**Original Version**

### *3.3 Description of Menu Function*

- *Maintains the link between the text interface and the graphical user interface. Provides the user with an option to run the program on the Graphical User Interface or the text Interface (). Only contains one function which is the menuFunction ( ).*

### *3.3.1 Interface Description*

*menuFunction( )*

- *This function does not return any values*
- *This does not have any parameters.*

***Algorithmic Model***

1. *Display options for (1)GUI or (2)Text Interface to the terminal.*
2. *Get input from the user.*
3. *Store user input into a integer variable called 'choice'*
4. *Error check to make sure 'choice' is equal to '1' or '2', if not equal to '1' or '2' prompt user for input again until correct input is entered, otherwise continue.*
5. *If choice is equal to 1 then branch of to the displayGui() function, otherwise continue.*
6. *If choice is equal to 2, then branch of to the textInterFaceOptions()function.*

***Local Data Structures***

- *None*

***Restrictions and Limitations***

- *None*

***Performance Issues***

- *None*

***Design Constraints***

- *None*

**Revised Version**

### 3.3.1 Interface Description

menuFunction( )

- Switched the order of option 1 and option 2.
- This function does not return any values
- This does not have any parameters.

**Algorithmic Model**

1. Display options for (1)Text Interface to the terminal or (2) GUI
2. Get input from the user.
3. Store user input into a integer variable called 'choice'
4. Error check to make sure 'choice' is equal to '1' or '2', if not equal to '1' or '2' prompt user for input again until correct input is entered, otherwise continue.
5. If choice is equal to 1 then branch of to the displayGui() function, otherwise continue.
6. If choice is equal to 2, then branch of to the textInterFaceOptions()function.

**Local Data Structures**

- None

**Restrictions and Limitations**

- None

**Performance Issues**

- None

**Design Constraints**

- None