Project 2A - Team1 Documentation

## Overview

Team 2 (ie The Wookie Workgroup) created a command line tool that computes the result of infix expressions (Project2A). The group consists of Daniel Mitchel, Joshua Neustrom, and Chen Wang.

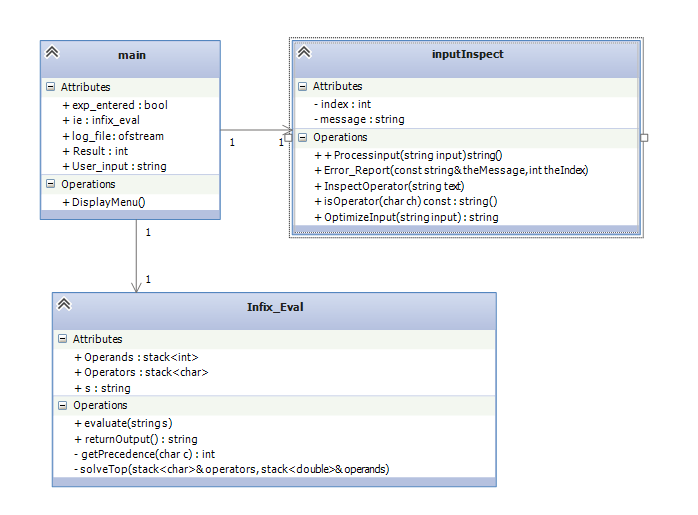
The report gives an overview of our solution including the following

1. Assumptions
2. UML Class Diagram
3. Github Project Link
4. Algorithm
5. Error Handling
6. Known Issues
7. Efficiency Analysis of Algorithms
8. References

## Assumptions

1. No variables used (can accept ints)
2. No history stored past the last expression entered
3. Spaces are cleaned from the user expression
4. Division involves ints only
5. Base ten number system
6. Only accepts () and no {} [] for setting precedence
7. Calculation or Syntax Error results in answer to expression being cleared
8. User can enter another expression if the last one entered results in an error
9. Expression Error Clears all previous user input (to purge any bad data)
10. Prefix increment and decrement only (no 1++, or 1--)
11. Wookies rule

## UML Class Diagram



## Github Site

<https://github.com/WookieWorkgroup/Project2/>

## Algorithm

1. User interface – continuous loop that displays a command line menu
2. Option 1 - Ask for Expression
   1. Store line
   2. Clean the string
      1. Optimize
         1. Eliminate Spaces
         2. Check for two operands in a row
         3. Check for Improper Paren position
      2. Inspect operators
         1. Check for paren error
         2. Check binary operator position
         3. Check for incomplete operator expression
         4. Check for two operators in a row
      3. Error messages include char position
      4. Return a cleaned string
   3. Compute the expression
      1. Make expression postfix by putting elements in the operand and operator stacks
         1. Read in digit and add to operand stack
         2. Read in operand
            1. If expression start in - at beginning then has digit, make the digit a negative
            2. If expression has – after operator, then make the digit after the - symbol a negative
            3. Add operands to stack

Check precedence

Higher precedence on bottom

* + 1. Compute the expression using postfix
       1. Pop operand
       2. Unary – Pop one operand and push result
       3. Binary – Pop two operands and push the result
       4. Once operator stack is empty, pop and return the result from the operands stack

1. Option 2 – Show the previous expression entered by user (if any)
2. Option 3 – Show the previous result (if any)
3. Option 4 – Clear – user input, results, ect
4. Option 5 – Exit (close console)

## Error Handling

1. Bad Input – Clean string returns an error and command line asks for input again
2. Logging – Log.txt contains record of actions completed to help troubleshoot
3. Try/Catch – for computation so error can be handled properly, errors during the calculation process include error code (pre calc error check also includes the char number)

## Known Issues

1. Division returns result an int. Possible to get divide by zero error due to ints that would not occur with floats

## Efficiency of Algorithms

1. Evaluate – O(n) or aprox T(2n) – one loop through string for convert to postfix, one loop through stacks to compute result
2. Inspect Input – O(n) or aprox T(3n) – Two loops to optimize and one loop to look for operators (loops are not nested)

## References

1. Evaluate algorithm - https://en.wikipedia.org/wiki/Shunting-yard\_algorithm
2. Method for eliminating user input that causes overflow - <http://stackoverflow.com/questions/3826281/how-do-i-make-a-c-program-that-filter-out-non-integers>
3. Post Fix, Syntax Error, and Infix to Postfix from class Lectures used as a starting reference
4. The Force