**CS 303: Project 1 Report**

evalInfix

* Break down of each function:
  + Constructors:
    - **evalInfix()**
      * *What does this function do?*
        + This function will assign a default value of “0” to “solution” as well as an empty string to “expression”. This is to initialize values of the class member variables.
      * *What is this function’s runtime?*
        + O(1)
    - **evalInfix(string infixExpression);**
      * *What does this function do?*
        + This function will assign a default value of “0” to “solution” as well as assigning the string passed in to “expression”. This is to initialize values of the class member variables.
      * *What is this function’s runtime?*
        + O(1)
  + Setters:
    - **void addOperator(string Operator);**
      * *What does this function do?*
        + This function appends “Operator” to “expression”. This is so the user can add operators to the current expression as they see fit.
      * *What is this function’s runtime?*
        + O(1)
    - **void addOperand(string Operand);**
      * *What does this function do?*
        + This function appends “Operand” to “expression”. This is so the user can add operands to the current expression as they see fit.
      * *What is this function’s runtime?*
        + O(1)
  + Public Functions:
    - **Int returnSolution();**
      * *What does this function do?*
        + Once the user has finished adding what they need to “expression”, this function will evaluate the current string stored in “expression”. It first starts by checking elements in “expression” and making sure they are either an ^, \*, /, +, -, or a number.
        + After elements have been parsed, this function goes through each element and calls “evalOperator” or “evalOperand” depending on if the element is an operator or operand (see corresponding function definitions for explanation on their operations).
        + After this, “returnSolution” then pops any remaining ‘(‘ and evaluates any other operands. Once this operation is completed, it returns the top element in the operand stack
      * *What is this function’s runtime?*
        + Best case: O(1)
        + Average: O(n)
        + Worst case: O(n^2)

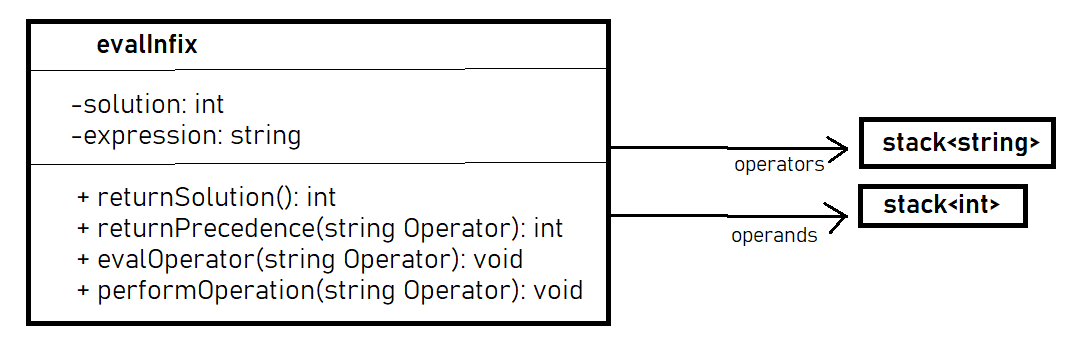
where ‘n’ is the number of characters in the expression (including spaces).

N^2 occurs if ‘returnSolution’ runs n times and calls ‘evalOperator’ which could run n times but is very unlikely to occur every iteration.

* + - **Int returnPrecedence(string Operator);**
      * *What does this function do?*
        + This function returns the corresponding precedence to the operators passed in. This is important as it allows “evalOperator” to determine in which order to evaluate expressions.
      * *What is this function’s runtime?*
        + O(1)
    - **Void evalOperator(string Operator);**
      * *What does this function do?*
        + Depending on the operator passed into it, this function will follow the same logic as described in “Infix to postfix conversion algorithm.pdf”. This function will mainly add operators to the operator stack, call the “performOperation” function to perform a given operation, and pop operators from the “operators” stack.
      * *What is this function’s runtime?*
        + Best case: O(1)
        + Worst case: O(n)

where ‘n’ represents number of operators until ‘(‘ is found, or until operator with less precedence is found.

* + - **Void evalOperand(int Operand);**
      * *What does this function do?*
        + This function will push elements to the ‘operands’ stack.
      * *What is this function’s runtime?*
        + O(1)
    - **Void performOperation(string Operator);**
      * *What does this function do?*
        + This function will pop the two top elements in the “operands” stack, and then perform an operation with the “Operator” passed in. It will then push the result to the “operands” stack.
      * *What is this function’s runtime?*
        + O(1)
* UML Diagram:



* Other Items to note:
  + evalInfix relies on the assumption that the user will enter data properly. Only characters that should be entered are ‘^, \*, /, +, -, \*spaces\*’ or any numerical value. This includes ‘0, 1, 2, 3, 4, 5, 6, 7, 8, 9, etc’.
  + There are also two main methods of creating expressions. First is to include the full expression when you create an object, and the other is to add to the expression as you go.