**Design Approach**

My design approach to evaluate the math expression is based on the Dijkstra’s “Shunting yard-algorithm” (<https://en.wikipedia.org/wiki/Shunting-yard_algorithm>) with the addition of business logic for unary and binary numbers.

The complete design contains below modules-

1. Math Expression Evaluator Application –

* The main file to start the execution of the program
* Accepts the math expression to be evaluated as an input from the user
* Outputs the final result to stdout

1. **Interface** – TokenizerInterface

* Initializes the Tokenizer with the source as an input math expression and an array of known Symbols
* It has two functions
  + GetNextToken() - Get the next available Token.
  + GetAllTokens() - Get all available Tokens (from the current position)

1. Token-

* A Token class to hold the information of the single token object.
* Returns the type-token it is.
* Returns value of the token as decimal, integer, or string

1. Tokenizer

* The Tokenizer separates the source string into individual Tokens (operators/symbols and integer/doubles).
* Tokenizer class initializes and provide the implementation for the interface methods
  + GetNextToken()
    - Reads next tokens from the string whether it is a digit, symbol or white space
    - Handles the case when the expression has an optional leading zero.

ReadOptionalDecimal() method append the optional leading zero while forming the digit.

* + GetAllToken()
    - Gets all the available token from the current position to be processed next.

1. Infix To Postfix Converter

* The InfixToPostfixConveter program converts a given math expression from infix notation to postfix notation.
* This class provides the implementation for the “Shunting-yard algorithm” with the additional business logic for the unary positive and negative number
* I have implemented the logic of shunting-yard algorithm described here <https://en.wikipedia.org/wiki/Shunting-yard_algorithm#The_algorithm_in_detail>
* **Pay attention to the implementation of a unary positive and negative number**

File Name: InfixToPostfixConverter.cs

protected string DoShuntingYardAlgorithm(TokenizerInterface tokenizer) {

//line number: 94 – 126

else if (IsOperator(token)){

1. check for a minus sign at the beginning of the input
   * + - set the value of current operation = m
2. check for a plus sign at the beginning of the input
   * + - set the value of current operation = P
3. check for minus sign direct after '(' or after another operator
   * + - set the value of current operation = m
4. check for minus sign direct after '(' or after another operator
   * + - set the value of current operation = P

}

}

* The output of this class is a string converted infix math expression into a postfix expression by the shunting-yard algorithm.

1. Postfix To Final Result

* Evaluates the result of a formula in postfix notation.
* Accepts the postfix string as input generated by the shunting-yard algorithm and evaluates the final result and outputs the decimal value.
* Uses stack store the operators and calculations
* **Pay attention to the implementation of a unary positive and negative number**

File Name: PostfixToFinalResult.cs

public decimal Evaluate(string postfixString)

{

//line number: 34 – 48

else if (IsUnaryOperator(entry))

{

case "m":

//if operator is a minus evaluate then

-1 \* topstack and push the result on the stack.

case "p":

// if the operator is a plus sign then

just push top stack back on the stack.

}

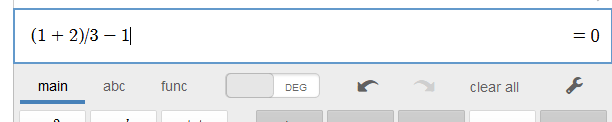
}

**Testing Approach**

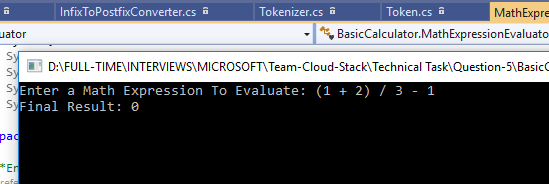
I have tested code with the following 2 approaches-

* + - 1. Manual Testing
      2. Testing using a Visual Studio Unit Test project
* **Manual Testing and Results (Some of the test cases with results)**

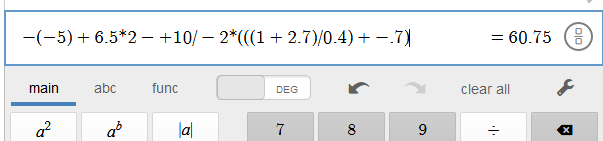
1. **Math Expression:** “(1 + 2) / 3 - 1”

**Expected Result -** 

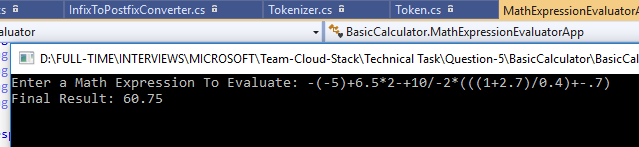
**Actual Result –**



1. **Math Expression – “**-(-5) + 6.5 \* 2 - +10 / -2 \* (((1 + 2.7) / 0.4) + -.7)**”**

**Expected Results -** 

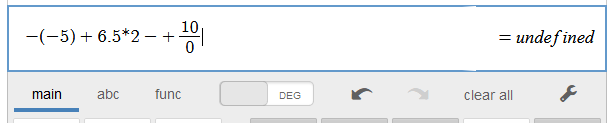
**Actual Result:**



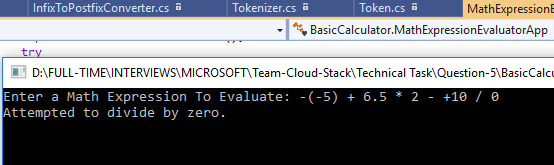
1. **Divide by Zero**

**Math Expression – “**-(-5) + 6.5 \* 2 - +10 / 0**”**

**Expected Results –**



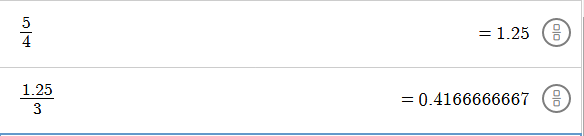
**Actual Result -**



1. **Left to right grouping precedence**

**Math Expression –** 5/4/3

**Expected Result**



**Actual Result**

