**Expression Evaluator Report Doc**

***Guntuka Venkatesh, 3rd year, CVR college of engineering***

***Data Structures used:***

***1. Character stack:*** *to convert the given infix expression to postfix.*

***2.Integer stack:*** *to evaluate the postfix expression and generate the result****.***

***Reason for the choice of Data Structures:***

* *The best way to evaluate an expression is by using a stack as it can implement operator precedence and other features like balanced parentheses effectively.*
* *A character stack has been used to convert the infix to postfix as the operators are characters.*
* *An integer stack has been used to evaluate the postfix expression as the result to be computed will be a number and it will avoid unnecessary character to integer conversions.*

***Algorithm used :***

***To generate postfix from given infix expression :***

*1. Print operands as they arrive.*

*2. If the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.*

*3. If the incoming symbol is a left parenthesis, push it on the stack.*

*4. If the incoming symbol is a right parenthesis, pop the stack and print the operators until you see a left parenthesis. Discard the pair of parentheses.*

*5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.*

*6. If the incoming symbol has equal precedence with the top of the stack, use association. If the association is left to right, pop and print the top of the stack and then push the incoming operator. If the association is right to left, push the incoming operator.*

*7. If the incoming symbol has lower precedence than the symbol on the top of the stack, pop the stack and print the top operator. Then test the incoming operator against the new top of stack.*

*8. At the end of the expression, pop and print all operators on the stack. (No parentheses should remain.)*

***To evaluate the postfix expression:***

*Initialize(Stack S)  
x = ReadToken(); // Read Token  
while(x)  
{  
 if ( x is Operand )  
 Push ( x ) Onto Stack S.  
  
 if ( x is Operator )  
 {  
 Operand2 = Pop(Stack S);  
 Operand2 = Pop(Stack S);  
 Evaluate (Operand1,Operand2,Operator x);  
 }  
  
 x = ReadNextToken(); // Read Token  
}*

***Specialities of the code:***

* *operator precedence implemented in the order : \*\*,/,\*,+,-*
* *additional functionality for parenthesis added.*
* *highly modular code.*
* *appropriate data structures used at each step.*