EXPERIMENT NO:-02

Date of Performance:

Date of Submission:

Aim : Convert an Infix expression to Postfix expression using stack ADT.

Theory:

Polish Notations:

Infix, postfix, and prefix notations are three different but equivalent notations of writing algebraic expressions. But before learning about prefix and postfix notations, We all are familiar with the infix notation of writing algebraic expressions. While writing an arithmetic expression using infix notation, the operator is placed in between the operands. For example, A+B; here, plus operator is placed between the two operands A and B. Although it is easy for us to write expressions using infix notation, computers find it difficult to parse as the computer needs a lot of information to evaluate the expression. Information is needed about operator precedence and associativity rules, and brackets which override these rules. So, computers work more efficiently with expressions written using prefix and postfix notations. *Postfix notation* was developed by Jan Łukasiewicz who was a Polish logician, mathematician, and philosopher. His aim was to develop a parenthesis-free prefix notation (also known as Polish notation) and a postfix notation, which is better known as Reverse Polish Notation or RPN. In postfix notation, as the name suggests, the operator is placed after the operands. For example, if an expression is written as A+B in infix notation, the same expression can be written as AB+ in postfix notation. The order of evaluation of a postfix expression is always from left to right. Even brackets cannot alter the order of evaluation.

The expression (A + B) * C can be written as:

[AB+]*C

AB+C*

To convert infix expression to postfix expression, we will use the stack data structure. By scanning the infix expression from left to right, when we will get any operand, simply add them to the postfix form, and for the operator and parenthesis, add them in the stack maintaining the precedence of them.

Algorithm : To convert Infix Expression to Postfix expression.

```
Step 1: Add  ) to the end of the infix expression

Step 2: Push ( on to the stack

Step 3: Repeat until each character in the infix notation is scanned

        IF a ( is encountered, push it on the stack

        IF an operand (whether a digit or a character) is encountered, add it

        postfix expression.

        IF a ) is encountered, then

                a. Repeatedly pop from stack and add it to the postfix expression until a
                   ( is encountered.

                b. Discard the ( . That is, remove the (from stack and do not
                   add it to the postfix expression
        IF an operator is encountered, then
              a.  Repeatedly pop from stack and add each operator (popped from the
                  stack) to the postfix expression which has the same precedence or a
                  higher precedence than 0.

              b. Push the operator 0 to the stack

        [END OF IF]

Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack
         is empty


Step 5: EXIT
```

Example:

(A – (B / C + (D % E * F) / G)* H)

Infix Character Scanned	Stack	Postfix Expression
(		
A	(	A
-	(	A
(	(- (	A
B	(- (	A B
/	(- (/	A B
C	(- (/	A B C
+	(- (+	A B C /
(	(- (+ (	A B C /
D	(- (+ (	A B C / D
%	(- (+ %	A B C / D
E	(- (+ %	A B C / D E
*	(- (+ % *	A B C / D E
F	(- (+ % *	A B C / D E F
)	(- (+	A B C / D E F * %
/	(- (+ /	A B C / D E F * %
G	(- (+/	A B C / D E F * % G
)	(-	A B C / D E F * % G / +
*	(- *	A B C / D E F * % G / +
H	(- *	A B C / D E F * % G / + H
)		A B C / D E F * % G / + H * -

Program:

Input & Output:

Conclusion:

Sign and Remark:

R1	R2	R3	R4	Total Marks	Signature
(3)	(5)	(4)	(3)	(15)	