A. Lab-9
Infix to Postfix using Stack.

B. Objective : To understand the use of stacks for infix expressions to postfix & prefix notations & to evaluate the resultant expressions.

C. Theory :

1. Infix : operators are between operands.
   eg: (a+b)*c
2. Postfix : Operators are ~~between~~ after operands.
   eg: a b + c *
3. Prefix : Operators are written before operands.
   eg: * + a b c

4. Stack - • Linear data structure that follows the LIFO (Last In First Out) principle
   • It is used to temporarily store operators & manage precedence during conversion.

D. Algorithm :

1. Infix → Postfix → T.C. : O(n)

Steps: - Initialize an empty stack for operators.
- Scan the infix expression from left to right.
- If scanned character is :
  • Operand : Add it to output string.
  • '(' : push it onto the stack.
  • ')' : pop from stack and add to output until '(' is encountered.
  • Operator: Precedence of top ≥ current opr. Pop & append it to output.

- Then push current operator onto Stack.
- Pop all remaining operators from Stack & append to output.

Result : Output String is the postfix expression.

2. Infix → Prefix → T.C : O(n)

Steps :
- Reverse the infix expression.
- Replace each '(' with ')' and vice versa.
- Convert the modified expression to postfix using the above algorithm.
- Reverse the resulting postfix expression.
- Result is prefix expression.

3. Postfix evaluation → T.C. : O(n)
- Initialize an empty stack.
- Scan expression from left to right.
- If scanned character is
  - Operand : Push it onto the stack.
  - Operator : Pop two elements from Stack, apply operator & push the result back.
- After the entire expression is scanned, value on the top of the stack is final result.

4. Prefix Evaluation → T.C. : O(n)
- Scan prefix expression from right to left.
- If scanned character is • operand : push it onto stack.
  • operator : pop two operands, apply operator, push result back