**TASK – 1.1:**

**1. Write algorithm for**

1. **Infix expression to prefix.**

Algorithm:

Step 1: First reverse the given expression

 Step 2: If the scanned character is an operand, put it into prefix expression.

 Step 3: If the scanned character is an operator and operator's stack is empty, push operator into operators' stack.

 Step 4: If the operator's stack is not empty, there may be following possibilities.

     If the precedence of scanned operator is greater than the top most operator of operator's stack, push this operator into operator 's stack.

     If the precedence of scanned operator is less than the top most operator of operator's stack, pop the operators from operator's stack untill we find a low precedence operator than the scanned character.

     If the precedence of scanned operator is equal then check the associativity of the operator. If associativity left to right then simply put into stack. If associativity right to left then pop the operators from stack until we find a low precedence operator.

     If the scanned character is opening round bracket ( '(' ), push it into operator's stack.

     If the scanned character is closing round bracket ( ')' ), pop out operators from operator's stack until we find an opening bracket ('(' ).

 Repeat Step 2,3 and 4 till expression has character

 Step 5: Now pop out all the remaining operators from the operator's stack and push into postfix expression.

 Step 6: Exit

**B.Infix expression to postfix.**

Algorithm:

Step 1: If the scanned character is an operand, put it into postfix expression.

Step 2: If the scanned character is an operator and operator's stack is empty, push operator into operators' stack.

Step 3: If the operator's stack is not empty, there may be following possibilities.

If the precedence of scanned operator is greater than the top most operator of operator's stack, push this operator into operator 's stack.

If the precedence of scanned operator is less than the top most operator of operator's stack, pop the operators from operator's stack until we find a low precedence operator than the scanned character.

If the precedence of scanned operator is equal then check the associativity of the operator. If associativity left to right then pop the operators from stack until we find a low precedence operator. If associativity right to left then simply put into stack.

If the scanned character is opening round bracket ( '(' ), push it into operator's stack.

If the scanned character is closing round bracket ( ')' ), pop out operators from operator's stack until we find an opening bracket ('(' ).

Repeat Step 1,2 and 3 till expression has character

Step 4: Now pop out all the remaining operators from the operator's stack and push into postfix expression.

Step 5: Exit