Stack

• Algorithm infix to postfix

- **Step 1** : Scan the Infix Expression from left to right.
- **Step 2** : If the scanned character is an operand, append it to Postfix string.
- **Step 3** : if operator
  - **Step 3.1** : If the precedence order of the scanned(incoming) operator is greater than the precedence order of the operator in the stack /or the stack is empty push it.
  - **Step 3.2 : Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator and copy poped to post fix, After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)**
- **Step 4** : If the scanned character is an '(' or '[' or '{', push it to the stack.
- **Step 5** : If the scanned character is an ')'or ']' or '}', pop the stack and and output it until a '(' or '[' or '{' respectively is encountered, and discard both the parenthesis.
- **Step 6** : Repeat steps 2-6 until infix expression is scanned.
- **Step 7** : Print the output
- **Step 8** : Pop and output from the stack until it is not empty.

algo in--->post(Amar style)

1.read infix start/left to right/end

2.till infix not over do

2.1 if read is (push on stack

2.2 if read is) pop everything till (is removed ,copy poped to postfix

(do not copy (,))

2.3 if read is operator :

if precedence(new op)>precedence(old on stack):

push on stack

else

pop everything till either condition satisfied/stack becomes empty, then push

2.4 if operand copy to post fix

3. copy left from stack to postfix

4.print postfix

• Algorithm Infix to prefix

- **Step 1** : Scan the Infix Expression from right to left.
- **Step 2** : If the scanned character is an operand, append to Prefix string.
- **Step 3** : Else,
  - **Step 3.1** : If the precedence order of the scanned(incoming) operator is greater than or equal to the precedence order of the operator in the stack or the stack is empty push it on stack.
  - **Step 3.2** : Else, Pop all the operators from the stack till either stack becomes empty or conditions satisfies. After doing that Push the scanned operator to the stack.
- **Step 4** : If the scanned character is an ‘)’or ‘]’ or ‘}’, push it to the stack.
- **Step 5** : If the scanned character is an ‘(‘ or ‘[‘ or ‘{‘, pop the stack and and output it until a ‘)‘ or ‘]‘ or ‘}‘ respectively is encountered, and discard both the parenthesis.
- **Step 6** : Repeat steps 2-6 until infix expression is scanned.
- **Step 7** : Print the output in reverse
- **Step 8** : Pop and output from the stack until it is not empty.

algo in--->pre(Amar style)

1.read infix right/end to left/start

2.till infix not over do

2.1 if read is) push on stack

2.2 if read is (pop everything till) is removed ,copy poped to prefix
(do not copy (,))

2.3 if read is operator :

if precedence(new op)>=precedence(old on stack):

push on stack

else

pop everything till either condition satisfied
Or stack becomes empty
then push

2.4 if operand copy to prefix

3. copy left from stack to prefix

4.print prefix in reverse manner

Algorithm to evaluate the postfix expression

1. Create a stack to store operands.
2. Scan the given expression from left to right.
3. If the scanned character is an operand,push it into the stack.
4. If scan char is operator, pop opnd2, opnd1 and perform the operation specified by the operator. Push the result into stack.
5. Repeat step 3 ,4 till all the characters are scanned.
6. When the expression is ended, the number in the stack is the final result.

- *2 3 1 * + 9 –*
- *100 ,200, + ,2 ,/, 5, *, 7, +*
- *123+*8-*

Algorithm to evaluate the prefix expression

1. Create a stack to store operands.
2. Scan one char at a time from right to left in string.
3. If scan char is operand, push it in stack.
4. If scan char is operator, pop opnd1, opnd2 and perform the operation specified by the operator. Push the result into stack.
5. Repeat step 2, 3 and 4 until input prefix strings end.
6. Pop opndstack and display, which is required value of given expression.

prefix expression

- *+9*26*
- *-+8/632*
- *-+7*45+20*