Infix to Postfix Conversion

Procedure for Postfix Conversion

1. Scan the Infix string from left to right.
2. Initialize an empty stack.
3. If the scanned character is an operand, add it to the Postfix string.
4. If the scanned character is an operator and if the stack is empty push the character to stack.
5. If the scanned character is an Operator and the stack is not empty, compare the precedence of the character with the element on top of the stack.
6. If top Stack has higher precedence over the scanned character pop the stack else push the scanned character to stack. Repeat this step until the stack is not empty and top Stack has precedence over the character.
7. Repeat 4 and 5 steps till all the characters are scanned.
8. After all characters are scanned, we have to add any character that the stack may have to the Postfix string.
9. If stack is not empty add top Stack to Postfix string and Pop the stack.
10. Repeat this step as long as stack is not empty.

Algorithm for Postfix Conversion

```
1. S:stack
2. while(more tokens)
3.   x<=next token
4.     if(x == operand)
5.       print x
6.     else
7.       while(precedence(x)<=precedence(top(s)))
8.         print(pop(s))
9.     push(s,x)
10.       while(! empty (s))
11.         print(pop(s))
```

Conversion To Postfix

EXAMPLE:

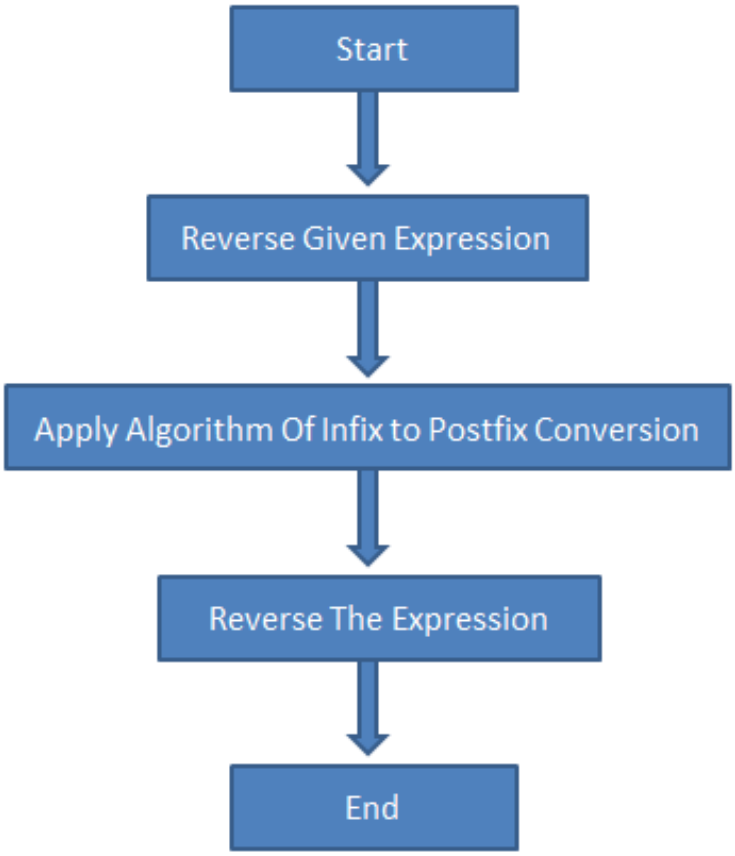
```
A+(B*C-(D/E-F)*G)*H
```

Stack	Input	Output
Empty	A+(B*C-(D/E-F)*G)*H	-
Empty	+(B*C-(D/E-F)*G)*H	A
+	(B*C-(D/E-F)*G)*H	A
+(	B*C-(D/E-F)*G)*H	A
+(	*C-(D/E-F)*G)*H	AB
+(*	C-(D/E-F)*G)*H	AB
+(*	-(D/E-F)*G)*H	ABC
+(-	(D/E-F)*G)*H	ABC*
+(-(	D/E-F)*G)*H	ABC*
+(-(	/E-F)*G)*H	ABC*D
+(-(/	E-F)*G)*H	ABC*D
+(-(/	-F)*G)*H	ABC*DE
+(-(-	F)*G)*H	ABC*DE/
+(-(-	F)*G)*H	ABC*DE/
+(-(-	)*G)*H	ABC*DE/F
+(-	*G)*H	ABC*DE/F-
+(-*	G)*H	ABC*DE/F-
+(-*	)*H	ABC*DE/F-G
+	*H	ABC*DE/F-G*-
+*	H	ABC*DE/F-G*-
+*	End	ABC*DE/F-G*-H
Empty	End	ABC*DE/F-G*-H*+

Infix to Prefix Conversion

Algorithm of Infix to Prefix

```
1. Step 1. Push ")" onto STACK, and add "(" to end of the A
2. Step 2. Scan A from right to left and repeat step 3 to 6 for each element
   of A until the STACK is empty
3. Step 3. If an operand is encountered add it to B
4. Step 4. If a right parenthesis is encountered push it onto STACK
5. Step 5. If an operator is encountered then:
6.     a. Repeatedly pop from STACK and add to B each operator (on the top of
    STACK) which has same
7.         or higher precedence than the operator.
8.      b. Add operator to STACK
9. Step 6. If left parenthesis is encontered then
10.        a. Repeatedly pop from the STACK and add to B (each operator on to
   p of stack until a left parenthesis is encounterd)
11.        b. Remove the left parenthesis
12.   Step 7. Exit
```



Infix to prefix conversion

Expression = **(A+B^C)*D+E^5**

Step 1. Reverse the infix expression.

5^E+D*)C^B+A(

Step 2. Make Every '(' as ')' and every ')' as '('

5^E+D*(C^B+A)

Step 3. Convert expression to postfix form.

```
A+(B*C-(D/E-F)*G)*H
```

Expression	Stack	Output	Comment
5^E+D*(C^B+A)	Empty	-	Initial
^E+D*(C^B+A)	Empty	5	Print
E+D*(C^B+A)	^	5	Push
+D*(C^B+A)	^	5E	Push
D*(C^B+A)	+	5E^	Pop And Push
*(C^B+A)	+	5E^D	Print
(C^B+A)	+*	5E^D	Push
C^B+A)	+*(	5E^D	Push
^B+A)	+*(	5E^DC	Print
B+A)	+*(^	5E^DC	Push
+A)	+*(^	5E^DCB	Print
A)	+*(+	5E^DCB^	Pop And Push
)	+*(+	5E^DCB^A	Print
End	+*	5E^DCB^A+	Pop Until '('
End	Empty	5E^DCB^A+*+	Pop Every element

Step 4. Reverse the output.

+*+A^BCD^E5

Result

+*+A^BCD^E5