161044028 - Data Structures and Algorithms HW4 - Q1

Q1:

Convert the infix expressions given below to prefix and postfix, then evaluate them. Show your work (how each token is processed and content of the stack afterwards) step by step.

i) A + ((B – C * D) / E) + F – G / H
ii) ! (A && ! ((B < C) | | (C > D))) | | (C < E)

Solution:

Which algorithm used:

Algorithm for Method convert

```
1. Initialize postfix to an empty StringBuilder
2. Initialize the operator stack to an empty stack.
3. while there are more tokens in the infix string.
4.      Get the next token.
5.      if the next token is an operand.
6.           Append it to postfix.
7.      else if the next token is an operator.
8.           Call processOperator to process the operator.
9.      else
10.          Indicate a syntax error.
11. Pop remaining operators off the operator stack and append them to postfix.
```

Algorithm for Method processOperator

```
1. if the operator stack is empty
2.      Push the current operator onto the stack.
   else
3.      Peek the operator stack and let topOp be the top operator.
4.      if the precedence of the current operator is greater than the precedence of topOp
5.           Push the current operator onto the stack.
   else
6.      while the stack is not empty and the precedence of the current operator is less than or equal to the precedence of topOp
7.           Pop topOp off the stack and append it to postfix.
8.           if the operator stack is not empty
9.                Peek the operator stack and let topOp be the top operator.
10.               Push the current operator onto the stack.
```

161044028 - Data Structures and Algorithms HW4 - Q1

General algorithm for infix to postfix:

1. Print operands as they arrive.

2. If the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.

3. If the incoming symbol is a left parenthesis, push it on the stack.

4. If the incoming symbol is a right parenthesis, pop the stack and print the operators until you see a left parenthesis. Discard the pair of parentheses.

5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.

6. If the incoming symbol has equal precedence with the top of the stack, use association. If the association is left to right, pop and print the top of the stack and then push the incoming operator. If the association is right to left, push the incoming operator.

7. If the incoming symbol has lower precedence than the symbol on the top of the stack, pop the stack and print the top operator. Then test the incoming operator against the new top of stack.

8. At the end of the expression, pop and print all operators on the stack. (No parentheses should remain.)

General algorithm for infix to prefix:

1.Reverse the given string-expression- (Also change the directions of parentheses).

2.Find the postfix of reversed string.

3.Reverse the expression.

161044028 - Data Structures and Algorithms HW4 - Q1

i) A + ((B – C * D) / E) + F – G / H

Let A=2, B=5, C=1, D=3, E=1, F=7, G=8, H=2. Result of this infix expression is:

2+ ((5 – 1 * 3) / 1) + 7 – 8 / 2 = 2 + ((2) / 1) + 7 – 4 = 7

Finding postfix: (In operator stack, left side is bottom, right side is top)

Input	Operator stack	Output
A		A
+	+	A
(	+(	A
(	+((	A
B	+((	AB
-	+((-	AB
C	+((-	ABC
*	+((-*	ABC
D	+((-*	ABCD
)	+(	ABCD*-
/	+(/	ABCD*-
E	+(/	ABCD*-E
)	+	ABCD*-E/
+	+	ABCD*-E/+
F	+	ABCD*-E/+F
-	-	ABCD*-E/+F+
G	-	ABCD*-E/+F+G
/	-/	ABCD*-E/+F+G
H	-/	ABCD*-E/+F+GH
	-	ABCD*-E/+F+GH/
		ABCD*-E/+F+GH/-

Postfix form of this expression is : ABCD*-E/+F+GH/-

Evaluating Postfix Expression: 2513*-1/+7+82/-

We go left to right through postfix expression(In the stack,left side is bottom, right side is top)

Input	Action	Stack
2	Push	2
5	Push	2,5
1	Push	2,5,1
3	Push	2,5,1,3
*	Pop(3,1) and 3*1=3 then push 3	2,5,3
-	Pop (5,3) and 5-3=2 then push 2	2,2
1	Push	2,2,1
/	Pop(2,1) and 2/1= 2 then push 2	2,2
+	Pop(2,2) and 2+2=4 then push 4	4
7	Push	4,7
+	Pop(7,4) and 7+4=11 then push 11	11
8	Push	11,8
2	Push	11,8,2
/	Pop(8,2) and 8/2=4 then push 4	11,4
-	Pop(11,4) and 11-4=7 then push 7	7

When we evaluate the postfix expression, same result found as evaluating infix form (both results are equal to 7).

161044028 - Data Structures and Algorithms HW4 - Q1

Finding prefix:

-Reverse string: H/G-F+(E/(D*C-B))+A

-Find postfix of this H/G-F+(E/(D*C-B))+A:

Input	Operator stack	Output
H		H
/	/	H
G	/	HG
-	-	HG/
F	-	HG/F
+	-+	HG/F
(	-+(	HG/F
E	-+(	HG/FE
/	-+(/	HG/FE
(	-+(/(	HG/FE
D	-+(/(	HG/FED
*	-+(/(*	HG/FED
C	-+(/(*	HG/FEDC
-	-+(/(-	HG/FEDC*
B	-+(/(-	HG/FEDC*B
)	-+(/	HG/FEDC*B-
)	-+	HG/FEDC*B-/
+	-++	HG/FEDC*B-/
A	-++	HG/FEDC*B-/A
		HG/FEDC*B-/A+
		HG/FEDC*B-/A++
		HG/FEDC*B-/A++-

HG/FEDC*B-/A++-

-Reverse string : -++A/-B*CDEF/GH

-Prefix form of this expression is: -++A/-B*CDEF/GH

Evaluating Prefix Expression: -++2/-5*1317/82

We go right to left through prefix expression(In the stack,left side is bottom, right side is top)

Input	Action	Stack
2	Push	2
8	Push	2,8
/	Pop(8,2) and 8/2=4 then push 4	4
7	Push	4,7
1	Push	4,7,1
3	Push	4,7,1,3
1	Push	4,7,1,3,1
*	Pop(1,3) and 1*3=3 then push 3	4,7,1,3
5	Push	4,7,1,3,5
-	Pop(5,3) and 5-3=2 then push 2	4,7,1,2
/	Pop(2,1) and 2/1=2 then push 2	4,7,2
2	Push	4,7,2,2
+	Pop(2,2) and 2+2=4 then push 4	4,7,4
+	Pop(4,7) and 4+7=11 then push 11	4,11
-	Pop(11,4) and 11-4=7 then push 7	7

After evaluating prefix expression we found 7(same result while evaluating postfix and infix version).

ii) ! (A && ! ((B < C) | | (C > D))) | | (C < E)

Let A=1 ,B=2, C=3, D=4, E=5. Result of this infix expression is:

!(1 && !((2<3) | | (3>4))) | | (3<5) = !(1 && !(1)) | | 1 = 1| |1 = 1

Finding postfix: (In operator stack, left side is bottom, right side is top of the stack)

Input	Operator stack	Output
!	!	
(	!(	
A	!(	A
&&	!(&&	A
!	!(&&!	A
(	!(&&!(	A
(	!(&&!((	A
B	!(&&!((	AB
<	! (&&!((<	AB
C	! (&&!((<	ABC
)	! (&&!(	ABC<
| |	! (&&!(| |	ABC<
(	!(&&!(| |(	ABC<
C	!(&&!(| |(	ABC<C
>	!(&&!(| |(>	ABC<C
D	!(&&!(| |(>	ABC<CD
)	!(&&!(| |	ABC<CD>
)	!(&&!	ABC<CD>| |
)	!	ABC<CD>| |!&&
| |	| |	ABC<CD>| |!&&!
(	| |(	ABC<CD>| |!&&!
C	| |(	ABC<CD>| |!&&!C
<	| |(<	ABC<CD>| |!&&!C
E	| |(<	ABC<CD>| |!&&!CE
)	| |	ABC<CD>| |!&&!CE<
		ABC<CD>| |!&&!CE<| |

Postfix form of this expression is : ABC<CD>| |!&&!CE<| |

161044028 - Data Structures and Algorithms HW4 - Q1

Evaluating Postfix Expression: 123<34>||!&&!35<||

We go left to right through postfix expression(In the stack,left side is bottom, right side is top)

Input	Action	Stack
1	Push	1
2	Push	1,2
3	Push	1,2,3
<	Pop(3,2) and 2<3=1 then pop 1	1,1
3	Push	1,1,3
4	Push	1,1,3,4
>	Pop(4,3) and 3>4=0 then pop 0	1,1,0
||	Pop(0,1) and 1||0=1 then pop 1	1,1
!	Pop(1) and !1=0 then push 0	1,0
&&	Pop(0,1) and 1&&0 =0 then push 0	0
!	Pop(0) and !0=1 then push 1	1
3	Push	1,3
5	Push	1,3,5
<	Pop(5,3) and 3<5=0 then push 0	1,0
||	Pop(0,1) and 0||1=1 then push 1	1

After evaluating the postfix form, same result found while evaluating the infix form.Both evaluation gave the result 1

Finding prefix:

-Reverse string: (E<C)||(((D>C) || (C<B)) ! && A)!
-Find postfix of this (E<C)||(((D>C) || (C<B)) ! && A)! :

Input	Operator stack	Output
(	(	
E	(	E
<	(<	E
C	(<	EC
)		EC<
||	||	EC<
(	||(	EC<
(	||((	EC<
(	||(((	EC<
D	||(((	EC<D
>	||(((>	EC<D
C	||(((>	EC<DC
)	||((	EC<DC>
||	||((||	EC<DC>
(	||((||(	EC<DC>
C	||((||(	EC<DC>C
<	||((||(<	EC<DC>C
B	||((||(<	EC<DC>CB
)	||((||	EC<DC>CB<
)	||(	EC<DC>CB<||
!	||(!	EC<DC>CB<||
&&	||(&&	EC<DC>CB<||!
A	||(&&	EC<DC>CB<||!A
)	||	EC<DC>CB<||!A&&
!	||!	EC<DC>CB<||!A&&
	||	EC<DC>CB<||!A&&!
		EC<DC>CB<||!A&&!||

-Reverse string : || ! && A ! || < B C > C D < C E
-Prefix form of this expression is: || ! && A ! || < B C > C D < C E

161044028 - Data Structures and Algorithms HW4 - Q1

Evaluating Prefix Expression: | | ! && 1 ! | | < 2 3 > 3 4 < 3 5

We go right to left through prefix expression(In the stack,left side is bottom, right side is top)

Input	Action	Stack
5	Push	5
3	Push	5,3
<	Pop(3,5) and 3<5=1 then push 1	1
4	Push	1,4
3	Push	1,4,3
>	Pop(3,4) and 3>4=0 then push 0	1,0
3	Push	1,0,3
2	Push	1,0,3,2
<	Pop(2,3) and 2<3=1 then push 1	1,0,1
| |	Pop(1,0) and 0| |1=1 then push 1	1,1
!	Pop(1) and !1=0 then push 0	1,0
1	Push	1,0,1
&&	Pop(1,0) and 1&&0=0 then push 0	1,0
!	Pop(0) and !0=1 then push 1	1,1
| |	Pop(1,1) and 1| |1=1 then push 1	1

After evaluating the prefix form, same result found as infix and postfix evaluations. All results are same (1).