

Stack Application

Evaluation of expression

1) Infix

Operand - operator – operand

A + B

10 + 20

2) Prefix- polish notation

Operator – operand – operand

+AB

3) Postfix – reverse polish notation – suffix form

Operand – operand – operator

AB+

()

/, *

+, -

Example:

INFIX -> POSTFIX

$P+(Q+R)-(Q*S/(P-Q))$

$P+ (Q+R) - (Q*S/(PQ-))$

$P+(QR+) - (Q*S/(PQ-))$

$P+(QR+) -(Q*S / PQ-)$

$P+(QR+) -(Q*S*PQ-/-)$

$(PQR++) - (Q*S*PQ-/-)$

$PQR++Q*S*PQ-/-$

Example:

$A+(B/C)+(D/A+(\underline{E-F})^G)$

$A+(B/C)+(D/A+(\underline{EF-})^G)$

$A+(\underline{B/C})+(D/A+(EF-G^))$

$A+(BC/)+(\underline{D/A}+(EF-G^))$

$A+(BC/)+(\underline{DA/} + (EF-G^))$

$\underline{A+(BC/)}+(DA/EF-G^+)$

$\underline{(ABC/+)} + \underline{(DA/EF-G^+)}$

$ABC/+DA/EF-G^++$

EXMPLE:

$A^*(B-C / (\underline{D+A}))$

$A^*(B-\underline{C} / (\underline{DA+}))$

$A^*(\underline{B - CDA+}/)$

$\underline{A^*(BCDA+/-)}$

$ABCD A+/-^*$

Example:

$A^*(B+C)-(\underline{C+D} / (\underline{A-B}))$

$A^*(B+C)-(\underline{C+D} / (\underline{AB-}))$

$A^*(B+C)-(\underline{C + DAB-}/)$

$$A^*(\underline{B+C})-(CDAB-/ +)$$

$$\underline{A^*(BC+)} - (CDAB-/ +)$$

$$\underline{ABC+^*} - (CDAB-/ +)$$

$$ABC+^*CDAB-/ + -$$

EXAMPLE:

$$X+(Y^*Z)+(P-(\underline{Q^*R})/Z)$$

$$X+(\underline{Y^*Z})+(P-(QR^*)/Z)$$

$$X+(YZ^*)+(P-(\underline{QR^*})/Z)$$

$$X+(YZ^*)+(P-(\underline{QR^*Z/}))$$

$$\underline{X+(YZ^*)}+(\underline{PQR^*Z/-})$$

$$(\underline{XYZ^*+})+(\underline{PQR^*Z/-})$$

$$XYZ^*+PQR^*Z/- +$$

Example :

$$(A+B)^*((\underline{C^*D})-E)^*F$$

$$(\underline{A+B})^*((\underline{CD^*})-E)^*F$$

$$(\underline{AB+})^*(\underline{CD^*E-})^*F$$

$$\underline{AB+CD^*E-^*^*F}$$

$$AB+CD^*E-^*F^*$$

Example:

$$A^*(\underline{B+D})/E-F^*(G+\underline{H/K})$$

$$A^*(BD+)/E-F^*(\underline{G+HK/})$$

$A*(BD+)/E-F*(GHK/+)$

$(ABD+*)/E-F*(GHK/+)$

$(ABD+*E/)-F*(GHK/+)$

$(ABD+*E/)-FGHK/+*$

ANS : $ABD+*E/FGHK/+*-$

EXAMPLE:

$A*(B+(C-A)/D)-C*D$

ANS : $ABCA-D/+*CD*-$

SOLVE THIS

Convert infix to prefix

$(a+b)*(c*d-e)*f$

$(+ab)*(*cd - e) *f$

$(+ab)*(-*cde)*f$

$*+ab-*cde _ f$

$**+ab-*cdef$

Example:

$A*(b+d)/e-f*(g+h/k)$

$A*(+bd)/e-f*(g + /hk)$

$A*(+bd)/e-f*(+g/hk)$

$(*a+bd)/e-f*(+g/hk)$

$/ *a+bde - f * (+g/hk)$

$/ *a+bde _ *f+g/hk$

Ans: $-/*a+bde*f+g/hk$

Que 1 : $(a+\underline{b/d})^{(\underline{e-f})*g}$

Ans: $(a+\underline{\quad}/\underline{bd})^{(\underline{-ef})*g}$

$(+a/bd)\underline{\quad}(*g-ef)$

$\wedge+a/bd*-efg$

Que 2 : $A*(\underline{b-c})+(\underline{a/c})-d$

$A*(-BC)+((\underline{/AC})-D)$

$\underline{A*(-BC)}+(-/ACD)$

$*A-BC \underline{\quad} + \underline{\quad} -/ACD$

$+*A-BC-/ACD$

Que 3: $x+(y/z-x)/(y+z)*x$

Convert it into prefix and postfix

Que : Translate the expression into infix notation and then evaluate it.

5,3,+,2,*,6,9,7,-,/,-

Solution :

Step 1: Convert it into infix notation

$(5+3) \underline{2,*,6,9,7,-,/,-}$

$((5+3)*2) \underline{6,9,7,-,/,-}$

$((5+3)*2)6,(\underline{9-7}),/,-$

$((5+3)*2)(6/(\underline{9-7}))-$

$((5+3)*2) - (6/(9-7))$

Step 2: solve this notation

$$((5+3)*2) - (6/(9-7))$$

$$((8)*2)-(6/(2))$$

$$(16)-(3)$$

Ans: 13

Que : convert postfix into infix

$$ABD+*E/FGHK/+*-$$

$$\text{Ans: } A(B+D) *E/FGHK/+*-$$

$$(A*(B+D)) E/FGHK/+*-$$

$$(A*(B+D)) E/FG(H/K)+*-$$

$$(A*B+D/E) \underline{FG(H/K)+*-}$$

$$(A*B+D/E) F(G+H/K)*-$$

$$(A*B+D/E) (F*G+H/K)-$$

$$(A*B+D/E) - (F*G+H/K)$$

	*
0	+

Top = -1

Algorithm Convert infix to postfix

Step 1: calculate the length of infix expression and assign it to variable L

Step 2: set $i=0$, $j=0$

Step 3: scan ith character of infix exp.

- (a) If '(' open bracket then push
- (b) If ')' close bracket then
Repeat while $s[\text{top}] \neq '('$ //example $\rightarrow (a+b) \Rightarrow ab+$
Postfix[j]=pop()
 $J=j+1$
Remove '('
- (c) If operator then
 - 1) Repeat while $s[\text{top}] \geq$ scan operator's priority // $+ \geq *$
Postfix[j]=pop()
 $J=j+1$
 - 2) Push operator
- (d) If operand then
Postfix[j]=infix[i]
 $J=j+1$

Step 4: $i=i+1$

Step 5: if $i < L$ then goto step 3

Step 6: make stack empty \rightarrow postfix[j]

$J=j+1$

Step7: postfix = answer

Example : $A + b * c - d$

$A + bc^* - d$

$Abc^* + _ d$

Ans: $Abc^* + d -$

Exp – infix	Push() - stack	Pop() - postfix
A		A
+	+	A
B	+	AB
*	+, *	AB
C	-	ABC*+
-	-	ABC*+
D	-	ABC*+D-

Symbol	Scanned	STACK	Postfix Expression	Description
1.		(Start
2.	A	(A	
3.	+	(+	A	
4.	((+(A	
5.	B	(+(AB	
6.	*	(+(*	AB	
7.	C	(+(*	ABC	
8.	-	(+(-	ABC*	'*' is at higher precedence than '-'
9.	((+(-(ABC*	
10.	D	(+(-(ABC*D	
11.	/	(+(-(/	ABC*D	
12.	E	(+(-(/	ABC*DE	
13.	^	(+(-(/^	ABC*DE	
14.	F	(+(-(/^	ABC*DEF	
15.)	(+(-	ABC*DEF^/	Pop from top on Stack , that's why '^' Come first
16.	*	(+(-*	ABC*DEF^/	
17.	G	(+(-*	ABC*DEF^/G	
18.)	(+	ABC*DEF^/G*-	Pop from top on Stack , that's why '^' Come first
19.	*	(+*	ABC*DEF^/G*-	
20.	H	(+*	ABC*DEF^/G*-H	
21.)	Empty	ABC*DEF^/G*-H*+	END

<u>Symbol.</u>	<u>Stack</u>	<u>Postfix.</u>	
(($*, / \rightarrow 2$
A		A	$+, - \rightarrow 1$
+	(+		- No two operators of same priority can stay together in the stack column.
B		AB	
/	(+ /		
C		ABC	
*	(+ *		
((+ * (ABC /	
D		ABC / D	
+	(+ * (+		
E		ABC / DE	
)	(+ * ()	ABC / DE +	
-	(+ * () -		
F		ABC / DE + * +	
)	(-)	ABC / DE + * + F	

Example: $(A+B/C*(D+E)-F)$

Algorithm Convert infix to Prefix

Step 1: calculate the length of infix expression and assign it to variable L

Step 2: set $i=0, j=0$

Step 3: Reverse the infix exp. // $(a+b)-c \rightarrow c-b+a($

Step 4: scan ith character of infix exp.

- If ')' close bracket then push
- If '(' open bracket then
Repeat while $s[\text{top}] \neq '('$ //example $\rightarrow (a+b) \Rightarrow$
 $\text{Prefix}[j] = \text{pop}()$
 $j = j + 1$
Remove ')'
 - If operator then

Repeat while $s[\text{top}] > \text{scan operator's priority} //$ $* > -$

Prefix[j]=pop()

J=j+1

Push operator

(d) If operand then

Prefix[j]=infix[i]

J=j+1

Step 5: $i=i+1$

Step 6: if $I < L$ then goto step 4

Step 7: make stack empty \rightarrow prefix[j]

J=j+1

Step 8: reverse the Exp that is our answered

$A + \underline{b * C} - D \rightarrow D - C * B + A$

$\underline{A + * BC} - D$

$+ A * BC \underline{- D}$

$- + A * BCD$

Exp – infix	Push() – stack	Pop() – prefix
D		D
-	-	D
C	$-(- > *)$	DC
*	-, *	DC
B	-, *	DCB
+	-, * ($* > +$), ($- > +$)	DCB*
A	-, +	DCB*A+-
	REVERSE:	-+A*BCD

$((A-B)+C*(D+E))-(F+G)$

$(A+(B*C))/(D-E)$

Exp – infix	Push() – stack	Pop() – prefix
))	
G		G
+)+	
F		GF
(GF+
-	-	
)	-)	
)	-))	
E		GF+E
+	-))+	
D	-)	GF+ED
(-)	GF+ED+
*	-)*	GF+ED+
C		GF+ED+C
+	-)+	GF+ED+C*
)	-)+)	
B		GF+ED+C*B
-	-)+)-	GF+ED+C*B
A		GF+ED+C*BA
(-)+	GF+ED+C*BA-
(-	GF+ED+C*BA-+
	MAKE STACK EMPTY	GF+ED+C*BA-+-
	REVERSE THE EXP	-+-AB*C+DE+FG

Task a:

We have infix expression in the form of:

$$((A-B)+C*(D+E))-(F+G)$$

Now reading expression from right to left and pushing operators into stack and variables to output stack

Input	Output_stack	Stack
)	EMPTY)
G	G)
+	G) +
F	GF) +
(GF +	EMPTY
-	GF +	-
)	GF +	-)
)	GF +	-))
E	GF + E	-))
+	GF + E	-)) +
D	GF + ED	-)) +
(GF + ED +	-)
*	GF + ED +	-) *
C	GF + ED + C	-) *
+	GF + ED + C *	-) +
)	GF + ED + C *	-) +)
B	GF + ED + C * B	-) +)
-	GF + ED + C * B	-) +) -
A	GF + ED + C * B A	-) +) -
(GF + ED + C * B A -	-) +
(GF + ED + C * B A - +	-
EMPTY	GF + ED + C * B A - + -	EMPTY

Infix Expression : $A + (B * C - (D / E ^ F) * G) * H$

Reverse the infix expression :

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

REVERSE ANSWER: $HGFE^D/*CB*-*A+$

Answer: $+A*-*BC*/D^EFGH$

Infix	Push(stack)	Pop(prefix)
H		H
*	*	
)	*)	
G		HG
*	*)*	
)	*)*)	
F		HGF
^	*)*)^	
E		HGFE
/	*)*)^ {^>/}	HGFE^
D	*)*)/	HGFE^D
(*)*	HGFE^D/
-	*)* {*>-}	HGFE^D/*
C	*)-	HGFE^D/*C
*	*)- {->*}	
B	*)-*	HGFE^D/*CB
(*	HGFE^D/*CB*-
+	* {*>+}	HGFE^D/*CB*-*
A	+	HGFE^D/*CB*-*A
		HGFE^D/*CB*-*A+

Evaluation of postfix operation:

Exp: 10,5,2,*,+

Exp	Op1	Operator	Op2	Result	stack
10					10
5					10,5
2					10,5,2
*	5	*	2	10	10,10
+	10	+	10	20	20
					Result = 20

Algorithm

Step 1: calculate the length of Postfix expression and assign it to variable L

Step 2: set i=0

Step 3: scan ith character of postfix expiration.

a) If operator then

Pop()->Op2

Pop()->op1

R= op1 operator op2

Push R

b) If operand then push()

Step 4: I = i+1

Step 5: if I<L then go to step 3

Step 6: pop and that is the answer

Evaluation of prefix : +3*25 -> reverse

Exp	Op1	Operator	Op2	Result	Stack
5					5
2					5 ,2
*	5	*	2	10	10
3					10 ,3
+	10	+	3	13	13
					Result - 13

Algorithm

Step 0: reverse the prefix expiration

Step 1: calculate the length of Postfix expression and assign it to variable L

Step 2: set i=0

Step 3: scan ith character of postfix expiration.

a) If operator then

Pop()->Op2

Pop()->op1

R= op1 operator op2

Push R

b) If operand then push()

Step 4: I = i+1

Step 5: if I<L then go to step 3

Step 6: pop and that is the answer

