INFIX TO PREFIX

Algorithm for the conversion of infix to prefix expression:

- First, reverse the infix expression given in the problem.
- Scan the expression from left to right.
- Whenever the operands arrive, print them.
- If the operator arrives and the stack is found to be empty, then simply push the operator into the stack.
- If the incoming operator has higher precedence than the TOP of the stack, push the incoming operator into the stack.
- If the incoming operator has the same precedence with a TOP of the stack, push the incoming operator into the stack.
- If the incoming operator has lower precedence than the TOP of the stack, pop, and print the top of the stack. Test the incoming operator against the top of the stack again and pop the operator from the stack till it finds the operator of a lower precedence or same precedence.
- If the incoming operator has the same precedence with the top of the stack and the incoming operator is ^, then pop the top of the stack till the condition is true. If the condition is not true, push the ^ operator.
- When we reach the end of the expression, pop, and print all the operators from the top of the stack.
- If the operator is ')', then push it into the stack.
- If the operator is '(', then pop all the operators from the stack till it finds) opening bracket in the stack.
- If the top of the stack is ')', push the operator on the stack.
- At the end, reverse the output.

INFIX EXPRESSION : a+b-(c^d)/e+f*(g^h)+i*k/l+m

First we have to **reverse** the infix string : **m+l/k*i+)h^g(*f+e/)d^c(-b+a**

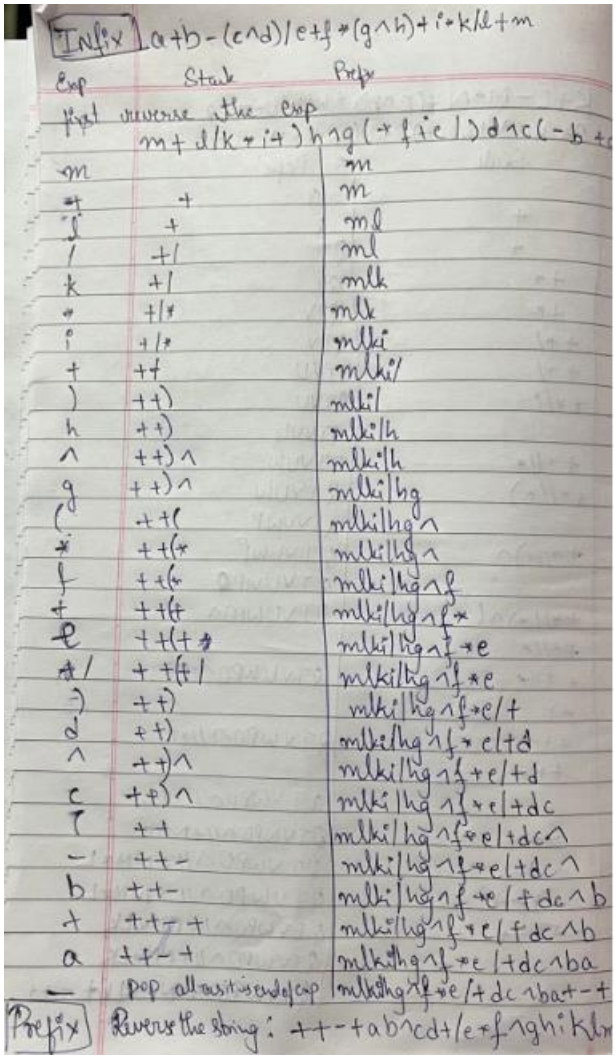
Expression	Stack	Prefix
m	-	m
+	+	m
l	+	ml
/	+/	ml
k	+/	mlk
*	+/*	mlk
i	+/*	mlki
+	++	mlki/
)	++)	mlki/
h	++)	mlki/h
^	++)^	mlki/h
g	++)^	mlki/hg
(	++(*	mlki/hg^
*	++(*	mlki/hg^
f	++(+	mlki/hg^f
+	++(+	mlki/hg^f*
e	++(+	mlki/hg^f*e
/	++(/	mlki/hg^f*e
)	++)	mlki/hg^f*e/+
d	++)	mlki/hg^f*e/+d
^	++)^	mlki/hg^f*e/+d
c	++)^	mlki/hg^f*e/+dc
(	++	mlki/hg^f*e/+dc^
-	++-	mlki/hg^f*e/+dc^
b	++-	mlki/hg^f*e/+dc^b
+	++-+	mlki/hg^f*e/+dc^b
a	++-+	mlki/hg^f*e/+dc^ba
End of expression	Pop all operators as it's end of the expression	mlki/hg^f*e/+dc^ba+-++

At the end, reverse the output to obtain prefix expression mlki/hg^f*e/+dc^ba+-++

Infix Expression : a+b-(c^d)/e+f*(g^h)+i*k/l+m

After conversion

Prefix Expression: ++-+ab^cd+/e*f^gh+iklm



INFIX TO POSTFIX

Algorithm for the conversion of infix to postfix expression:

1. Print the operand as they arrive.
2. If the stack is empty or contains a left parenthesis on top, push the incoming operator on to the stack.
3. If the incoming symbol is '(', push it on to the stack.
4. If the incoming symbol is ')', pop the stack and print the operators until the left parenthesis is found.
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 lower precedence than the top of the stack, pop and print the top of the stack. Then test the incoming operator against the new top of the stack.
7. If the incoming operator has the same precedence with the top of the stack then use the associativity rules. If the associativity is from left to right then pop and print the top of the stack then push the incoming operator. If the associativity is from right to left then push the incoming operator.
8. At the end of the expression, pop and print all the operators of the stack.

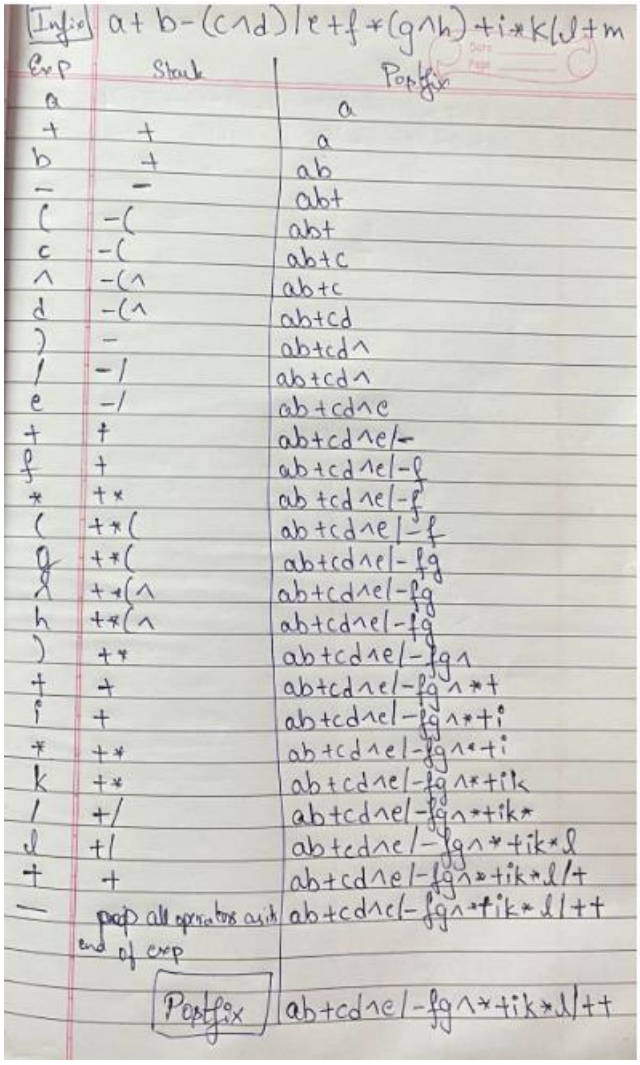
INFIX EXPRESSION : a+b-(c^d)/e+f*(g^h)+i*k/l+m

Expression	Stack	Postfix
a		a
+	+	a
b	+	ab
-	-	ab+
(	-(	ab+
c	-(	ab+c
^	-(^	ab+c
d	-(^	ab+cd
)	-	ab+cd^
/	-/	ab+cd^
e	-/	ab+cd^e
+	+	ab+cd^e/-
f	+	ab+cd^e/-f
*	+*	ab+cd^e/-f
(	+*(	ab+cd^e/-f
g	+*(	ab+cd^e/-fg
^	+*(^	ab+cd^e/-fg
h	+*(^	ab+cd^e/-fg
)	+*	ab+cd^e/-fg^
+	+	ab+cd^e/-fg^*+
i	+	ab+cd^e/-fg^*+i
*	+*	ab+cd^e/-fg^*+i
k	+*	ab+cd^e/-fg^*+ik
/	+/	ab+cd^e/-fg^*+ik*
l	+/	ab+cd^e/-fg^*+ik*l
+	+	ab+cd^e/-fg^*+ik*l/+
End of expression	Pop all operators as it's end of the expression	ab+cd^e/-fg^*+ik*l/++

Infix Expression : a+b-(c^d)/e+f*(g^h)+i*k/l+m

After conversion

Postfix Expression: ab+cd^e/-fg^*+ik*l/++



Infix: a+b-(c^d)/e+f*(g^h)+i*k/l+m

Exp	Stack	Postfix
a		a
+	+	a
b	+	ab
-	-	ab+
(	-(	ab+
c	-(	ab+c
^	-(^	ab+c
d	-(^	ab+cd
)	-	ab+cd^
/	-/	ab+cd^
e	-/	ab+cd^e
+	+	ab+cd^e/-
f	+	ab+cd^e/-f
*	+*	ab+cd^e/-f
(	+*(	ab+cd^e/-f
g	+*(	ab+cd^e/-fg
^	+*(^	ab+cd^e/-fg
h	+*(^	ab+cd^e/-fg
)	+*	ab+cd^e/-fg^
+	+	ab+cd^e/-fg^*+
i	+	ab+cd^e/-fg^*+i
*	+*	ab+cd^e/-fg^*+i
k	+*	ab+cd^e/-fg^*+ik
/	+/	ab+cd^e/-fg^*+ik*
l	+/	ab+cd^e/-fg^*+ik*l
+	+	ab+cd^e/-fg^*+ik*l/+
—	pop all operators until end of exp	ab+cd^e/-fg^*+ik*l/++

Postfix: ab+cd^e/-fg^*+ik*l/++