Infix to Postfix using stack

O(n)

Ex (a+b*c)

↳ (a+(b*c))

⇒ a+(bc*)

⇒ abc*+

BODMAS

(a+b) + d

Infix to Postfix using stack

^ * + - ↑ Higher Precedence

Operator	Associativity
^	Right to left
* , /	left to right
+ , −	left to right

High Precedence ↑ Low Precedence

Closed Bracket ( )

we consider its precedence is low

∴ ^ → Exponent

Algorithm

Infix to Postfix

1. Create empty stack, st
2. Traverse the expression from left to right and do following with every character x
3. if x is :-
   a) operand : print it
   b) left paranthesis : **Push to** st
   c) Right Paranthesis : Pop from st, until left Parthesis is found, output the popped character
   d) operator : if st is empty, push x to st.
      else compare with st top.
   → if x has higher precedence (than st top). **Push** to st.
   → Lower precedence, pop st top, and output until a higher precedence operator is found. **then** push x to st.
   → **Equal** precedence, use associativity.
4. pop and output Everything from st.

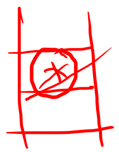
try to think in term of **precedence**

- top of stack is always to be **higher precedence**
- if top of stack and current operator has same precedence then we check the Associativity

( )

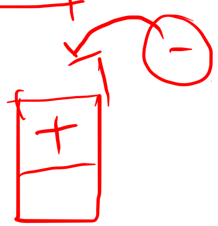
(2+3)

2+3*5-6



+ / -

* ^



- Associativity

+ -	(L to R)
* /	

means, operator present at left side has higher precedence than Right side.

^ (R to L)

1. Create empty stack, St
2. Traverse the expression from left to right and do following with every character x
3. if x is :-
   a) operand : print it
   b) left parenthesis : **Push to** St
   c) Right Parenthesis : Pop from St, until left Parenthesis is found, output the popped character
   d) operator : if St is empty, Push x to St. else compare with St top.
      - → if x has higher precedence (than St top), **Push** to St.
      - → Lower precedence, pop St top, and output until a higher precedence operator is found. **then** Push x to St.
      - → **Equal** precedence, use associativity.
4. pop and output Everything from St.

Ex-1 I/p: a+b*c → a+(b*c) ⇒ a+bc* ⇒ abc*+

Input symbol	stack	Result
a	⌊	a
+	+	a
b	+	ab
*	* / +	ab
c	* / +	abc
Pop elem	⌊	abc*+

ans ↑

1. Create empty stack, st
2. Traverse the expression from left to right and do following with every character x
3. if x is :-
   a) operand : print it
   b) left paranthesis : Push to st
   c) Right Paranthesis : Pop from st, until left Paranthesis is found, output the popped character
   d) operator : if st is empty, Push x to st. else compare with st top.
   → if x has higher precedence (than st top). Push to st.
   → Lower precedence, pop st top, and output until a higher precedence operator is found. then push x to st.
   → Equal precedence, use associativity.
4. Pop and output Everything from st.

Ex-2 I/p: (a+b) * d

o/p: (ab+) * d

ab+d*

Input symbol	stack	Result
(	(	
a	(	a
+	(+	a
b	(+	ab
)		ab+
*	*	ab+
d	*	ab+d
pop		ab+d*

1. Create empty stack, st
2. Traverse the expression from left to right and do following with every character x
3. if x is :-
   a) operand : print it
   b) left paranthesis : Push to st
   c) Right Paranthesis : Pop from st, until left Parthesis is found, output the popped character
   d) operator : if st is empty, Push x to st.
      else compare with st top.
   → if x has higher precedence (than st top). Push to st.
   → Lower precedence, pop st top, and output until a higher precedence operator is found. then push x to st.
   → Equal precedence, use associativity.
4. pop and output Everything from st.

^ x / + c

Ex3 I/p: a*b/c → a*(bc/) ⇒ abc/*

dp

Input symbol	stack	Result
a	⌊ ⌋	a
*	*	a
b	*	ab
/	/ *	ab
c	/ *	abc
pop		abc/*

① Create empty stack, st

② Traverse the expression from left to right and do following with every character x

③ if x is :-

a) operand : print it

b) left paranthesis : **Push to** st

c) Right Paranthesis : Pop from st, until left Parthesis is found, output the popped character

d) operator : if st is empty, Push x to st.
else compare with st top.

→ if x has higher precedence (than st top). **Push** to st.

→ Lower precedence, pop st top, and output until a higher precedence operator is found. **then** push x to st.

→ **Equal** precedence, use associativity.

④ pop and output Everything from st.

Ex-4 I/p: a+b/c−d*e

o/p:

Input symbol	stack	Result
a		a
+	+	a
b	+	ab
/	+/	ab
c	+/	abc
−	−	abc/+
d	−	abc/+d
*	−*	abc/+d
e	−*	abc/+de
		abc/+de*−

o/p →

1) Create empty stack, st
2) Traverse the expression from left to right and do following with every character x
3) if x is :-
   a) operand : print it
   b) left paranthesis : Push to st
   c) Right Paranthesis : Pop from st, until left Paranthesis is found, output the popped character
   d) operator : if st is empty, Push x to st.
      else compare with st top.
   → if x has higher precedence (than st top). Push to st.
   → Lower precedence, pop st top, and output until a higher precedence operator is found. then push x to st.
   → Equal precedence, use associativity.
4) pop and output Everything from st.