Infix to postfix -
Rules

Create 1 array to store postfix expr, 1 stack

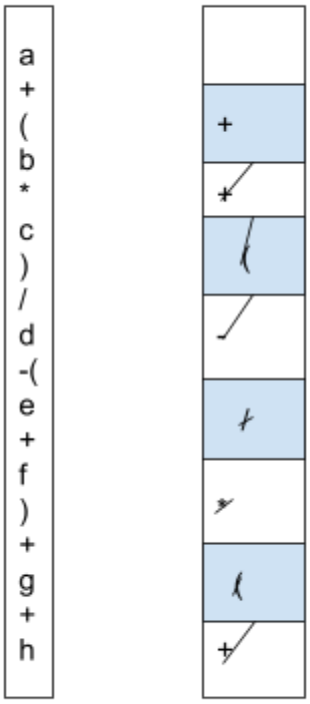
1. Write expression vertically and take one by one character
2. If it is operand add elt in postfix array
3. If it is operator then push one by one operators in stack based on precedence
    a. Rules to add operator in stack
        i. If arriving operator has higher precedence than inner one operator of stack then push new operator in stack over inner one
        ii. If precedence is lower/equal than inner operator then pop all elements from stack and add in to post array and then push new operator
        iii. If any opening brackets come push into stack as it has more precedence
        iv. If any operator comes and in stack at top if any opening bracket is there then add the operator directly on opening bracket and then after for second operator check precedence with inner operator and follow rule i)
        v. After expression is finished if any operator remaining in stack then pop all element and add in to post array one by one
        vi. That is resultant postfix expression

Infix to prefix-
Steps-

1. Reverse infix expression
2. Then exchange brackets with opening to closing and closing to opening
3. Then follow steps to convert infix to postfix conversion then
    a. While converting into in to post for pre only one rule is changed that If precedence is lower than inner operator then pop all elements from stack and add in to post array and then push new operator (if new operator is equal precedence then pop inner operator)
4. Reverse the result of postfix expression
5. The result is prefix conversion

a+(b*c)/d-(e+f)+g+h

abc*d/+ef+-g+h+



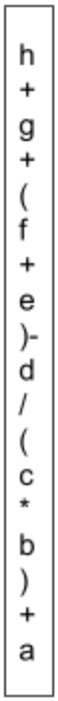
) -) +

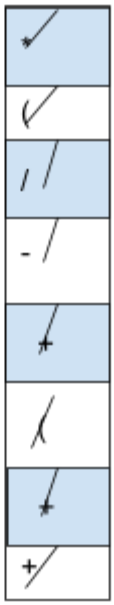
Operations for each operator
+ Push
(Push
* Push
For→)
* Pop
(Pop
/ Push
For → -
/ pop
+ pop
- push
(push
+ Push
For →)
+ pop
+ push
For → +
+ pop
+ push
+ pop

a+(b*c)/d-(e+f)+g+h
+++a-/*bcd+efgh

ghfe+++dcb*/-a+ →h

Reverse → h+g+)f+e(-d/)c*b(+a ⇒ h+g+(f+e)-d/(c*b)+a





) -) +

Operations for each operator
+ Push
+Push
(Push
+ push
For→)
+ Pop
+ pop
+ pop
- Push
/ Push
(push
* push
For →)
* pop
(pop
For →+
/ pop
- pop
+ push
+ pop