30/09/2020

Application of stacks:-

1) infix to Postfix
2) Balancing of ~~pa~~ paranthesis
3) Evaluation of an expression.

1) Infix expression:-

operators , operands.

Expression:- A+B

A*B+C*D

A+B+C+D.

infix:- <operand> operator <operand>

2) Postfix Expression

<operand1> <operand2> <operator>

A B+

3) Prefix Expression:-

operator <operand1> <operand2>

⇒ Conversion of infix to Postfix Expression:-

Steps:-

ⓐ If the character is left paranthesis, Push to the stack

ⓑ " " is an operand, Add it to the Postfix expression.

ⓒ If the character is an operator, check whether the stack is Empty.

① If the stack is empty, Push operator on to the stack

ii) If the stack is not empty, check the priority of the operator.

i) If the priority of the operator > operator present at the 'top' of the stack, then push operator into the top of the stack.

ii) If the priority of operator ≤ operator present at top of the stack, then pop the operator from stack and ADD to postfix expression expression goto step (i)

d) If the character is right parenthesis, then pop all the ~~characters~~ operators from the stack until it reaches left parenthesis and add to postfix expression.

e) After reading all characters if stack is not empty, then pop and add to post-fix expression.

Examples:-

a - (b * c - d) / e

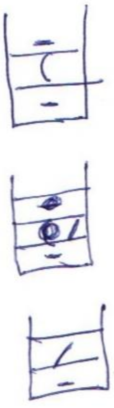
character	stack	postfix
a	-	a
-	-	a
(	(-	a
b	(-	ab
*	* (-	ab
c	* (-	abc
-	- (-	abc*

d

)

/

e



abc*d

abc*d -

abc*d -

abc*de/-

EX2:- a/ b-c+d*e-a*c : ab/c-de*+ac*-

EX3:- (a/ (b-c+d)) *(e-a)*c

Read greater push

Less - pop then push

(a / (b - (+ d)) *

greater push

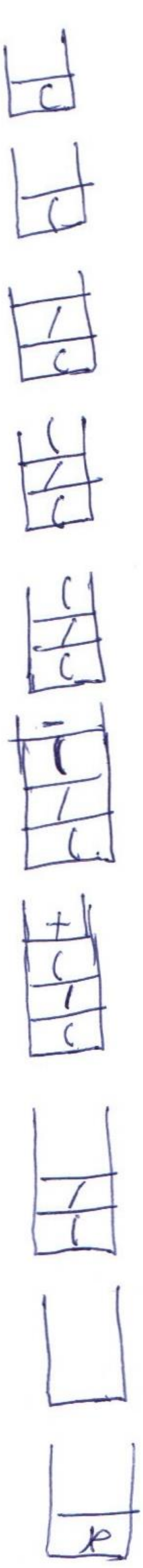
Pop - Push+

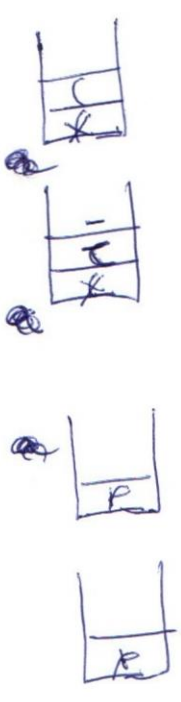
equal pop

pop until '('

pop until '('

(e - a) * e





a | abc-d+/e

a | abc-d+/e

a | abc-d+/ea

abc-d+/ea-

ab | abc-d+/ea- *

abc-d+/ea -*C*

ab

abc

abc-d

abc-d+

abc-d+/

abc-d+/

abc-d+e/ea- *c*