INFIX TO POSTFIX CONVERSION:

PRIORITY

1. * / — 1st
2. + — — 2nd

RULES:

1) STACK ⇒ operator
2) ARRAY ⇒ Alphabets

Higher → First
Smaller → Next.

3) Smaller/equal priority operator cannot be placed upon a larger/equal priority operator, if the condition occur larger/equal priority operator in stack must be "popped" to array & then get inserted in Stack.

4) Brackets can be pushed into Stack, but once when you encounter a close bracket all the operators between an open and close bracket are popped out into the array. and here the brackets are eliminated.

a*(b+c).

Infix Notation
operator b/w its operands

Prefix Notation
Polish Notation
operator is written before operands *a+bc

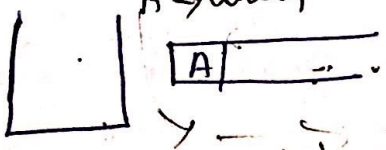
Postfix Notation
Reverse Polish Notation
operator is written after its operands
abc+*

PROBLEMS:

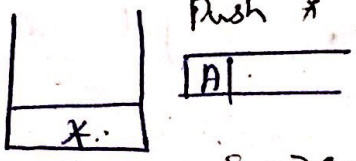
1. $A * (B+(C+D) * (E+F)/G) * H \rightarrow ABCD+EF+*G/+*H*$
2. $A + [(B+C) * (D+E) * F] / G \rightarrow ABC+DE+*F*G/+$
3. $A * (B+D) / E - F - (G + H / K) \rightarrow ABD+*E/F-GHK/+-$
4. $((A-B) * (D/E)) / (F*G*H) \rightarrow AB-DE/*FG*H*/$

1. $A * (B+(C+D) * (E+F)/G) * H.$

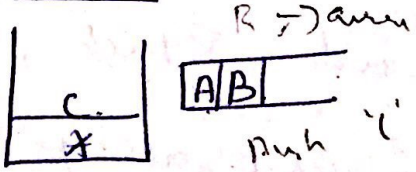
a) A → array



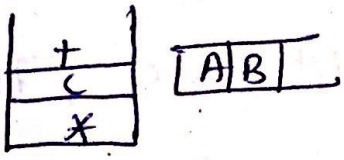
b)



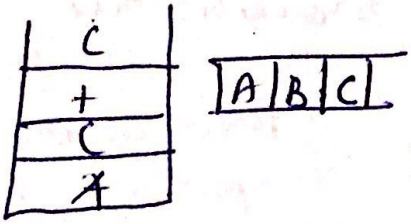
c) B → array



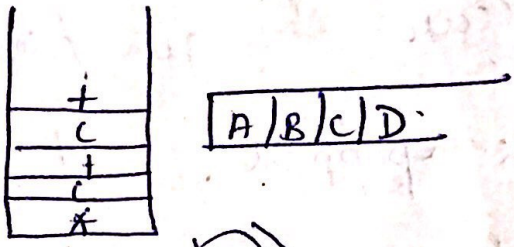
d)



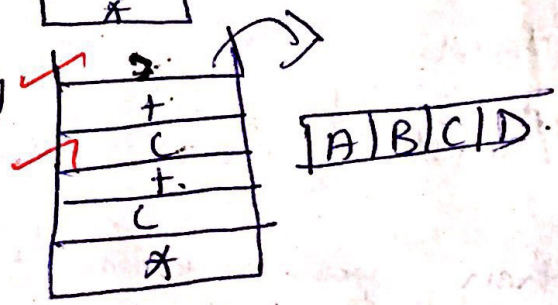
e)



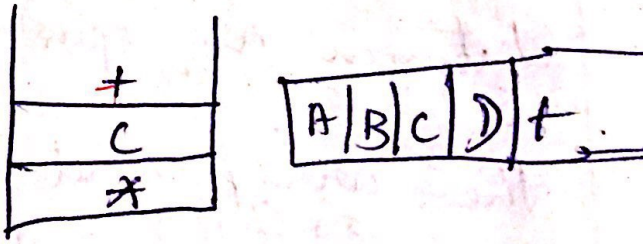
f)



g)



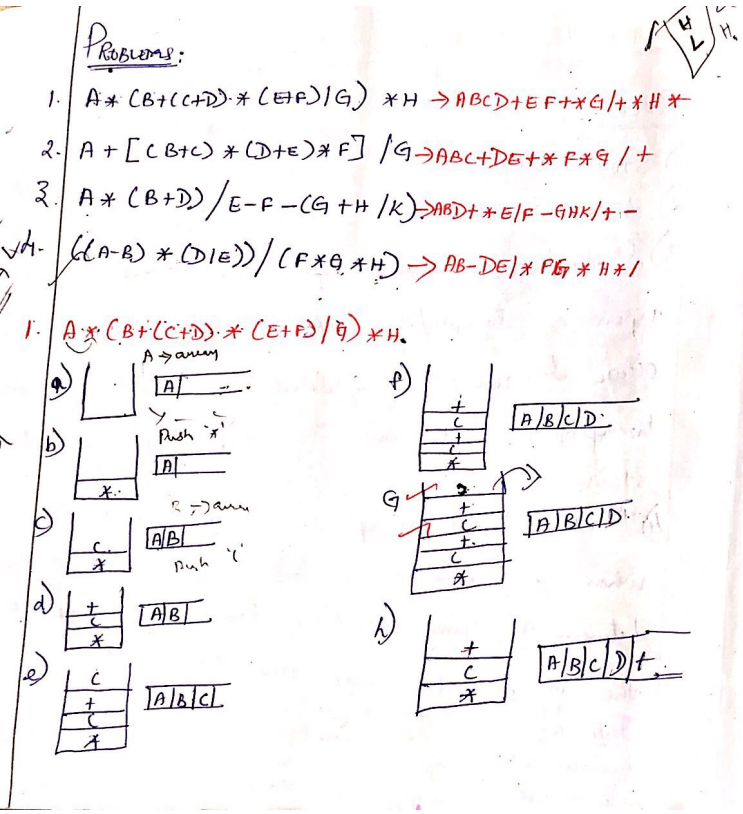
h)

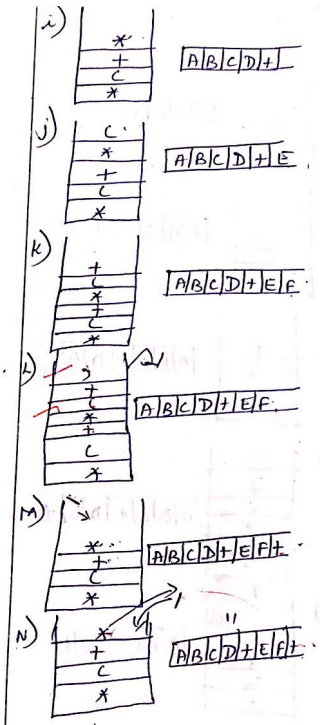


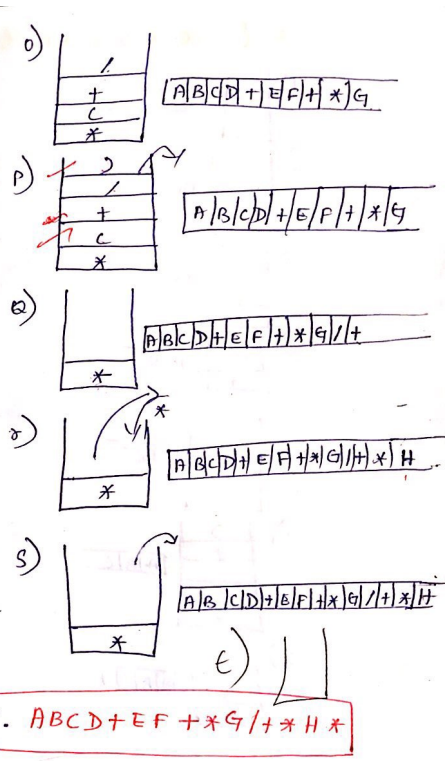
Problems:

1. $A * (B + (C + D) * (E + F) / G) * H \rightarrow ABCD+EF+*G/+*H*$
2. $A + [(B + C) * (D + E) * F] / G \rightarrow ABC+DE+*F*G/+$
3. $A * (B + D) / E - F - (G + H / K) \rightarrow ABD+*E/F-GHK/+-$
4. $((A - B) * (D / E)) / (F * G * H) \rightarrow AB-DE/*FG**H*/$

1. $A * (B + (C + D) * (E + F) / G) * H.$

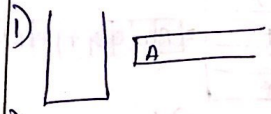


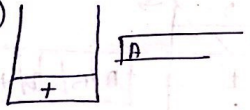


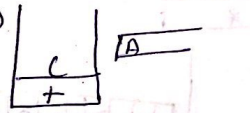


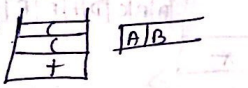
$\therefore ABCD+EF+*G/+*H*$

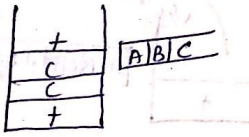
2) A + [(B + C) * (D + E) * F) / G

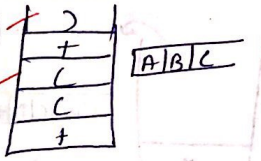
1) 

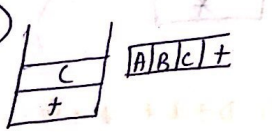
2) 

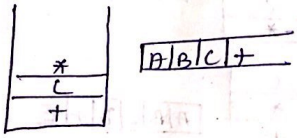
3) 

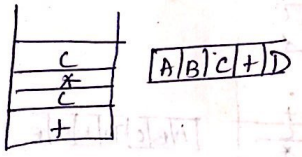
4) 

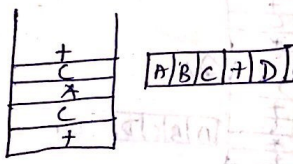
5) 

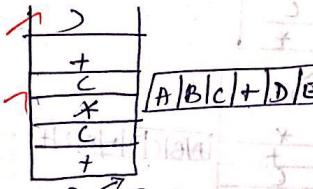
6) 

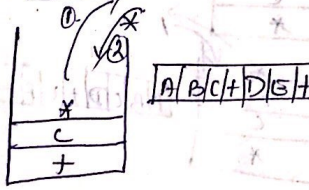
7) 

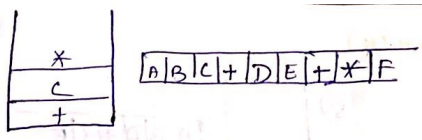
8) 

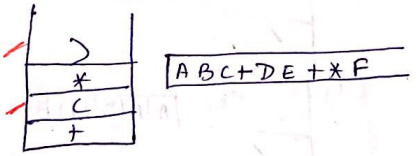
9) 

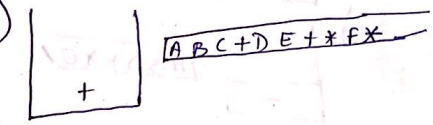
10) 

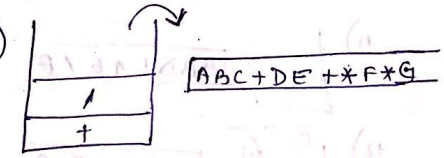
11) 

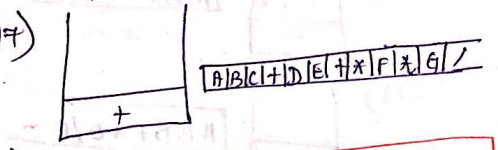
12) 

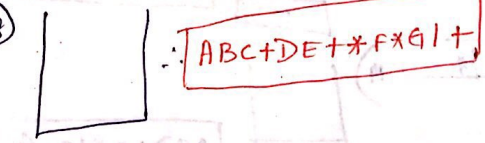
13) 

14) 

15) 

16) 

17) 

18) 

II

EVALUATING ARITHMETIC EXPRESSION

STEPS:

1→ Stack → number

2) ARRAY → when an operator comes (or) gets its turn from the stack the operator is put blw the last two no, and the values are calculated & push into stack again.

PROBLEMS:

1) AB + C − BA + C ^ − A=1 ; B=2 ; C=3. (−27)

2) 623 + − 382 / + * 2 ^ 3 + (52)

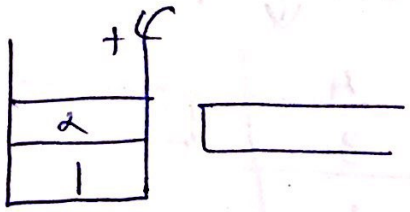
3) 12 + 34 * +

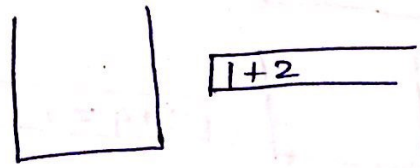
1) A B + C − B A + C ^ −

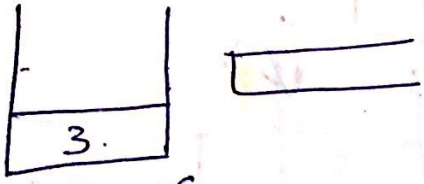
1 2 + 3 − 2 1 + 3 ^ −

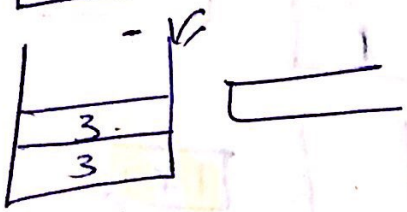
* POSTFIX EVALUATION:

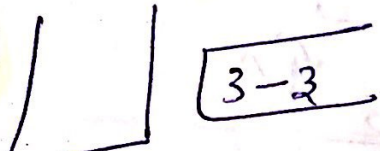
→ When the elements are popped, 1st element comes to right hand side of operator & 2nd to left hand side

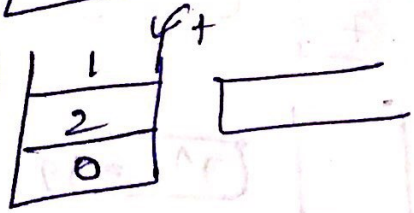
1) 

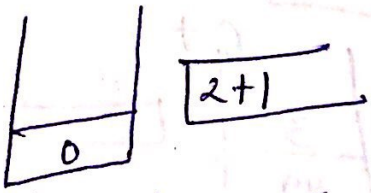
2) 

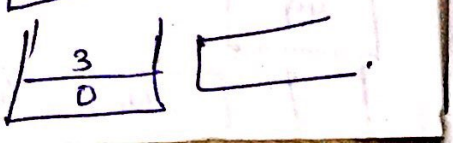
3) 

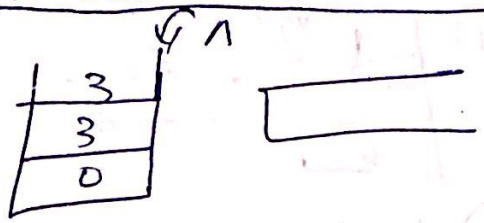
4) 

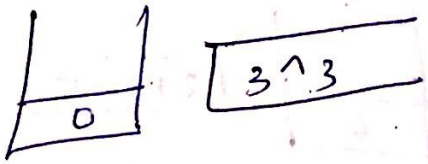
5) 

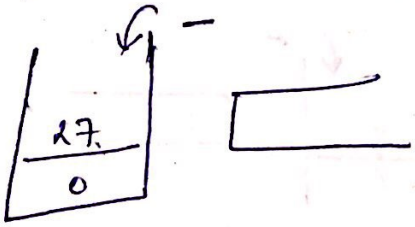
6) 

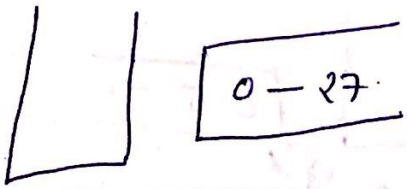
7) 

8) 

9) 

10) 

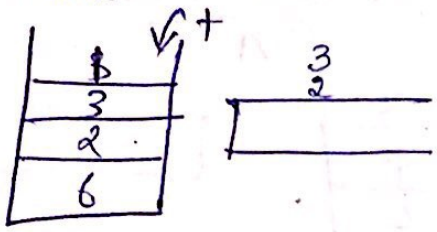
11) 

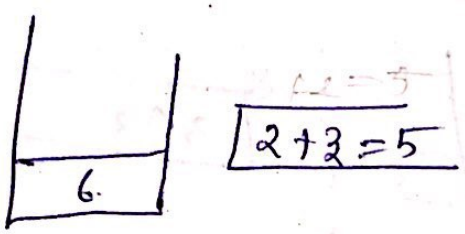
12) 

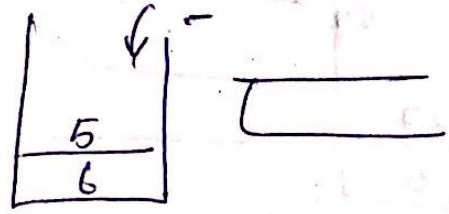
∴ −27

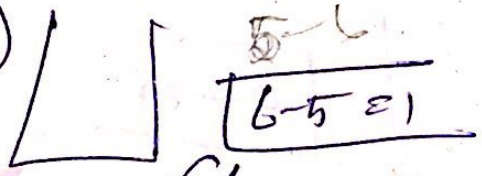
2) 623 + − 382 / + * 2 ^ 3 +

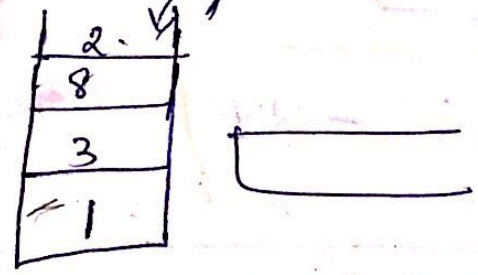
2+2

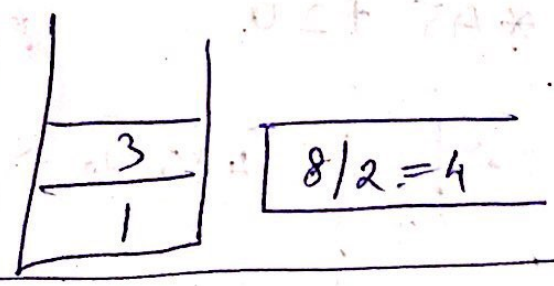
1) 

2) 

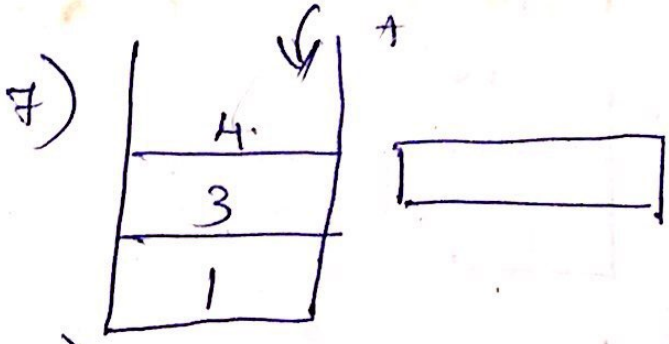
3) 

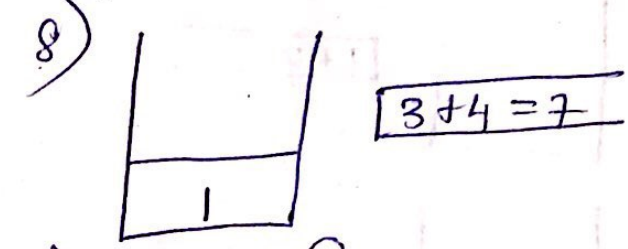
4) 

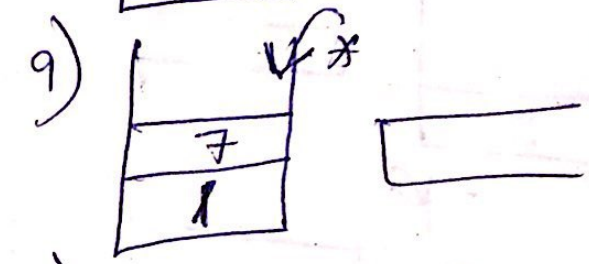
5) 

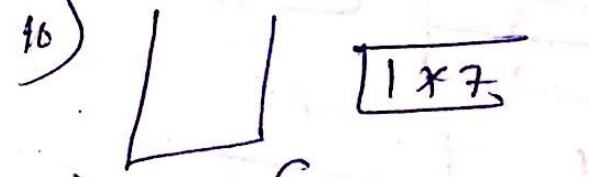
6) 

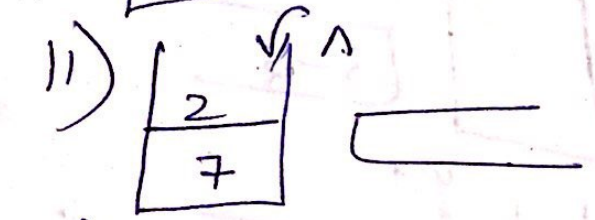
∴ 52.

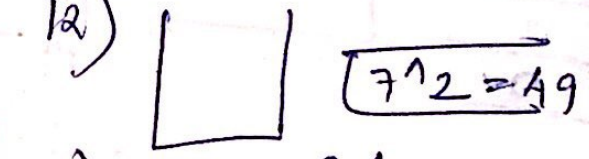
7) 

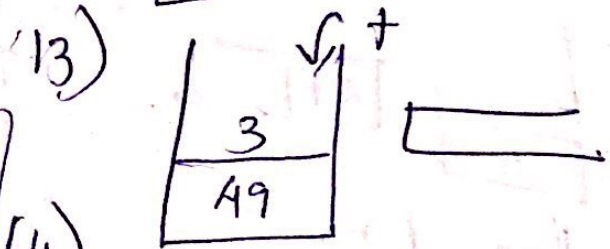
8) 

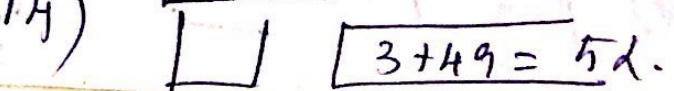
9) 

10) 

11) 

12) 

13) 

14) 

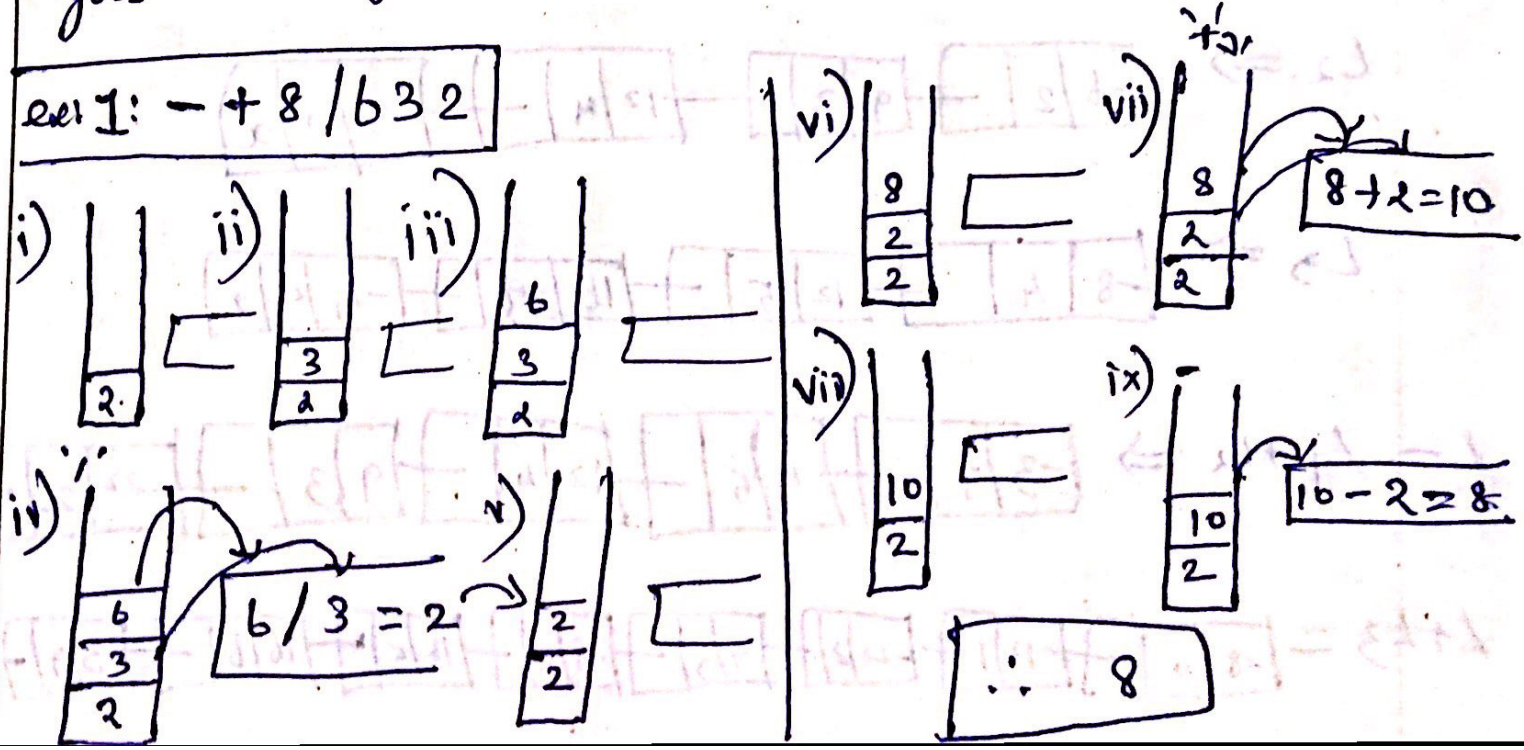
3) PREFIX EVALUATION:

1) Same Rules as used in Postfix Evaluation

2) But, while reading the input expression read it from right to left direction

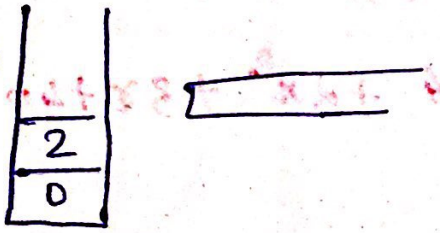
3) When elements are popped out when an operator comes, the 1st element comes to left hand side of operator & Second element goes to right hand side of operator

Ex: 1: − + 8 / 6 3 2

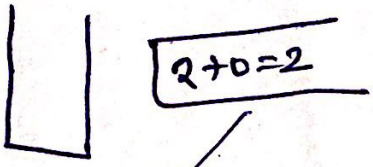


2) − + 7 * 45 + 20

1)

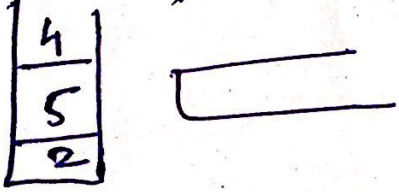


2) +

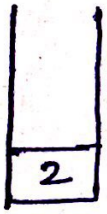


2+0=2

3)



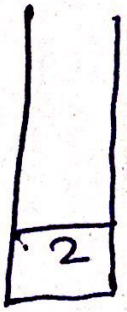
4) *



4 * 5 = 20

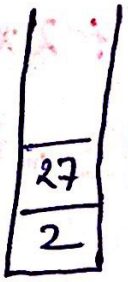
5)

6) + 2



7 + 20 = 27

7)



8) − 2

27 − 2 = 25

∴ 25