STACK

Date / / /

STACK = LIFO = Last In - First Out

Example = Stack of Plates

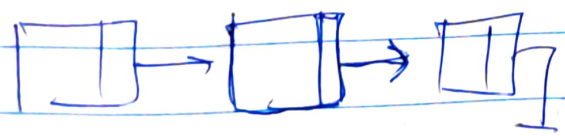
Abstract Data Type of STACK :—

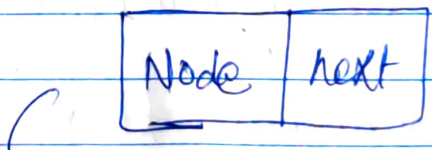
Data := 1. Space for elements to store
2. Top pointer

Operations := 1. push(x) 2. pop()
3. peek(index) 4. stacktop() 5. isEmpty()
6. isFull()

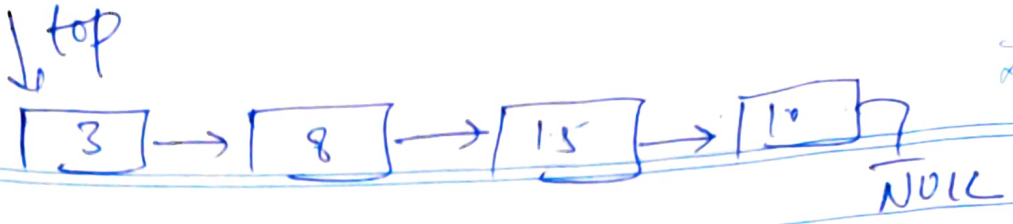
Physical Data Structure :- (i) Array (ii) Linked List. (LL)

STACK Using LL :=






```
Struct Node {
int data;
Struct Node * next;
}
```



Empty condition if (top == NULL)

```
void push(int x) {
Node *t = new Node
if (t == NULL)
  cout << "Stack Overflow";
else {
  t->data = x;
  t->next = top;
  top = t;
}}
```

```
void pop()
if (top != NULL) {
  Node *p;
  p = top
  top = top->next
  delete p;
}}
```

Application of STACK :-

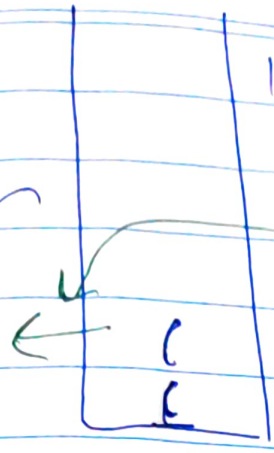
Paranthesis Matching :- For every opening paranthesis there should be closing paranthesis.

((a+b) * (c-d)) = ✓

((a+b) * (c-d) = ✗ Missing.

Date / / /

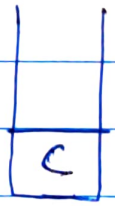
1. If opening or closing symbol then response else ignore (a, b + etc)



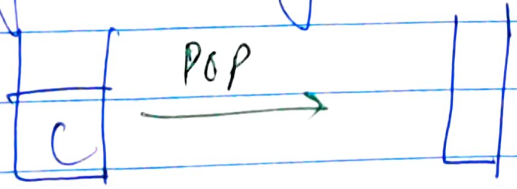
2. If we ignore other symbol we get closing bracket whenever closing bracket [POP]

3. After * symbol we get ')' closing then again push.

4. Got closing bracket pop the symbol.



5. Again closing bracket POP.

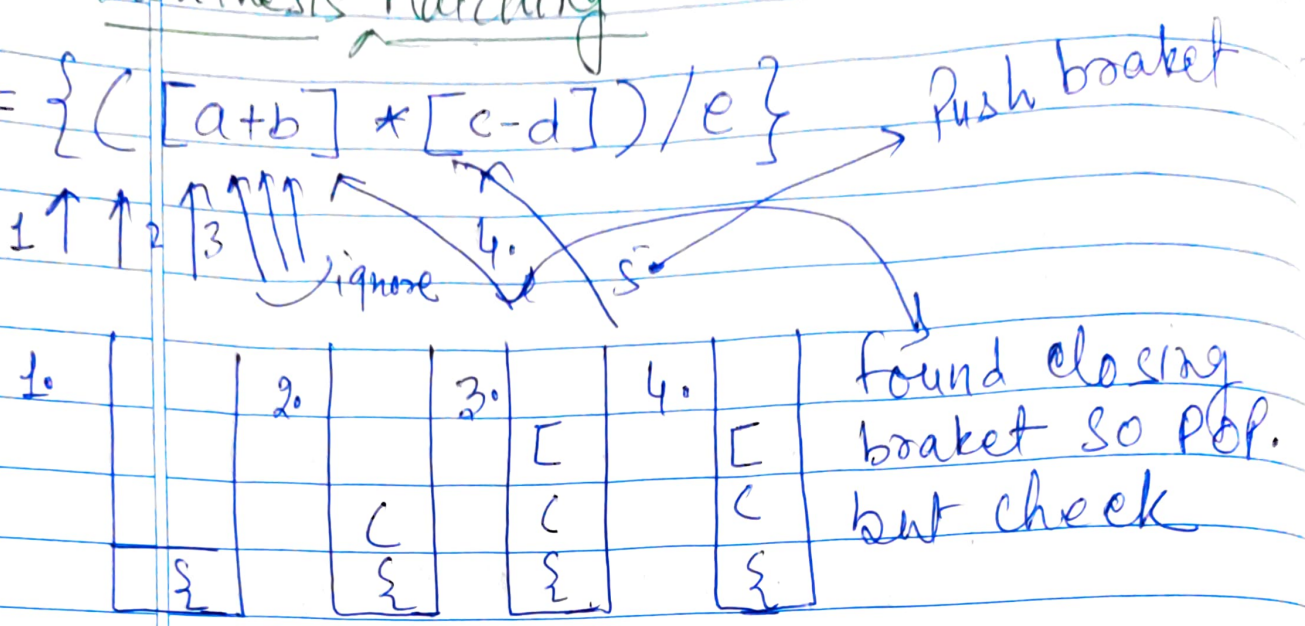


→ Stack is empty at the END & complete string is parsed.

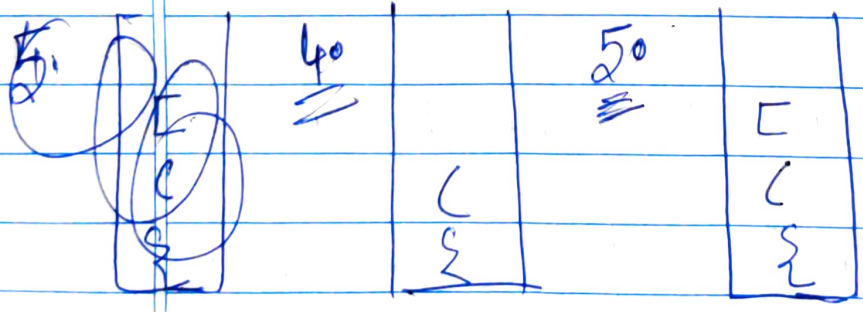
Date: / /

Paranthesis Matching :-

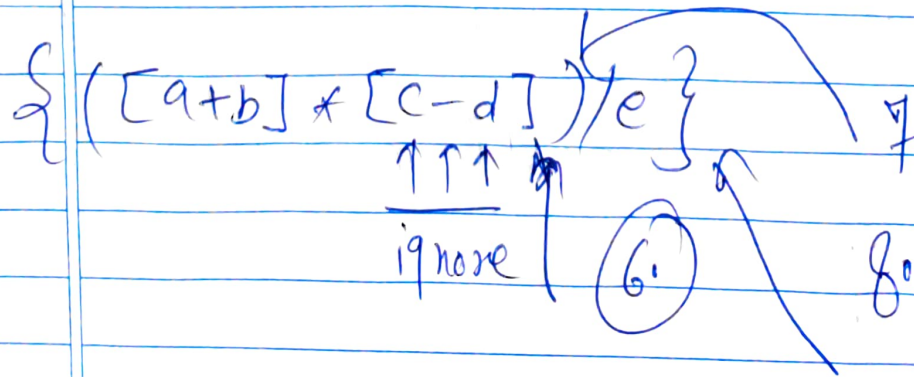
exp = {([a+b] * [c-d])/e}



the pop bracket should match means ([])({})(()) so current closing braket & pop bracket should match. which is correct in this case.

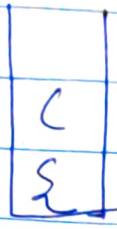


exp: {([a+b] * [c-d])/e}

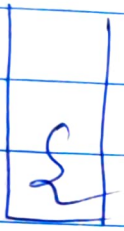


Date / /

6. Pop from stack & check the matching



7. Pop from stack & check matching.



8. Pop from stack & check the matching.

exp end stack is empty.

Infix to Postfix Conversation :-

1. Infix: a+b → Operand Operator Operand
2. Prefix: +ab → Operator Operand Operand
3. Postfix: ab+ → Operan Operand Operator

Date: / /

Infix to Postfix :-

Symbol Precedence

+, -	1
*, /	2
( )	3

← Higher Val Higher Precedence.

exp = a + b * c

Precedence is used for PARANTHESIS. Not for execution purpose.

→ Lets paranthesis

a + b * c → a + (b * c) → (a + (b * c))

high ↑ ↑ It is paranthesis.

Exp → (a + (b * c))

exp → a + b + c * d → a + b + (*cd)

↑ ↑

→ Both of them have equall precedence hence Left to Right.

→ (+ab) + (*cd) →

→ + +ab * cd → PREFIX FORM

→ Saneway Postfix. → POSTFIX

→ ab + cd + +

Infix To Postfix STACK

Date: / / /

Sym	Pre	Asso
+, -	1	L-R
*, /	2	L-R

Exp ↓

a+b*c-d/e

S.No	Sym	Stack	Postfix
1.	a	——	a
2.	+	+	a
3.	b		ab
4.	*	+*	
5.	c		abc
6.1	-	+	abc*
6.2		——	abc*+
7.		'-'	ab*+
8.	d	-	ab*+d

1. Operand Stack is empty give it to POSTFIX.

2. Operator anything in stack Equal to or Higher than '+' operator NO. Push to stack.

3. Operand symbol send it to POSTFIX Exp

4. Operator is there anything in stack on top equall to or less than current operator NO. i.e ('+' is there in stack with lower precedence). So push current Operator. in STACK.

5. Operand send it to post fix.

6.1 Operator '-' it's precedence 1. Stack top has '*' precedence higher i.e 2. POP OUT TOP & send to POSTFIX.

Date / /

6.2 Again chek STACK TOP '+' with precedence '1' which is equall '-' precedence. Hence POP '+' also.

7. Nothing inside Stack Push '-' in STACK.

8. Operand 'd' send to Postfix.

S.NO	Sym	STACK	Postfix
9.	/	-/	abc*+d
10.	e		abc*+de
11.			abc*+de/- ← stack Empty

9. Operator '/' Stack top operator '-' with precedence 1. '/' current precedence 2. Push to Stack.

10. Operand 'e' send to Postfix.

11. Exp. ended empty STACK & send to POSTFIX.

[PROGRAM FOR INFIX TO POST FIX]