Todays Content:

- → Stacks Basics
- → Double Character trouble
- → Sort Stack using Only Stacks
- → Expression Evaluation
  - a) Infix → postfix : Idea
  - b) Evaluate postfix : Code

Stacks: It uses $\underbrace{\text{LIFO}}_{\text{last in first out}}$ or $\underbrace{\text{FILO}}_{\text{first in last out}}$

Functnolities:

: push(x)

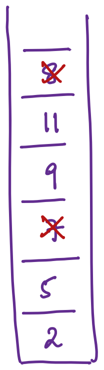
: pop()

: top()

: size()

Ex: 2 5 7 pop() top() 9 11 8 pop() top()

$\underline{\text{pop()}} \to 7$ $\underline{\text{top()}} \to 5$ $\underline{\text{pop()}} \to 8$ $\underline{\text{top()}} \to 11$



$\boxed{\text{Note: Push / pop() happens at same side}}$

Stack library:

$$\underbrace{\text{stack}}_{\text{stack}}<\underbrace{\text{int}}_{\text{type of stack}}>\ \text{st};$$

st; → st.push() → st.pop() → st.top() → st.size()

Note: A single operation takes O(1)

Stack using dynamic arrays:

list<int> l;

Ex: 2 5 7 $\underset{7}{\text{pop()}}$ $\underset{5}{\text{top()}}$ 9 11 8 $\underset{8}{\text{pop()}}$ $\underset{11}{\text{top()}}$

0	1	2	3	4
2	5	9	11	~~8~~

Implementation using dynamic arrays

push(x) : Insert x at last index : O(1)

pop() : Delete ele at last index : O(1)

top() : Return ele at last index : O(1)

size() : Return size : O(1)

Q) Double Character Trouble

Given a string s, Remove equal pair of adjacent characters
Return the string without adjacent duplicates

Ex1: a $\cancel{b}$ $\cancel{b}$ d → ad

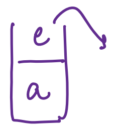
Ex2: a $\cancel{b}$ $\cancel{c}$ $\cancel{c}$ $\cancel{b}$ d e → a d e

Ex3: a $\cancel{b}$ $\cancel{b}$ b e → a b e

Ex4: a $\cancel{d}$ $\cancel{x}$ $\boxed{\cancel{b}\ \cancel{b}}$ $\cancel{x}$ $\cancel{c}$ $\boxed{\cancel{a}\ \cancel{a}}$ $\cancel{c}$ $\cancel{d}$ e → ae

↳ Idea: insert each character in stack TC: O(N) SC: O(N)

: if new character & top of stack are same pop the from stack



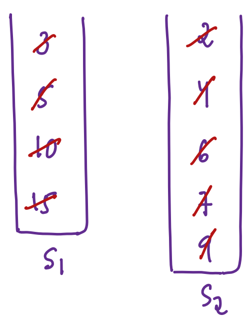
: e a extract char from stack

Note: Reverse final data we get from stack

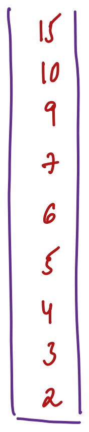
↳ Code TODO

2Q) Given 2 sorted stacks merge them into one sorted stack

Ex1:



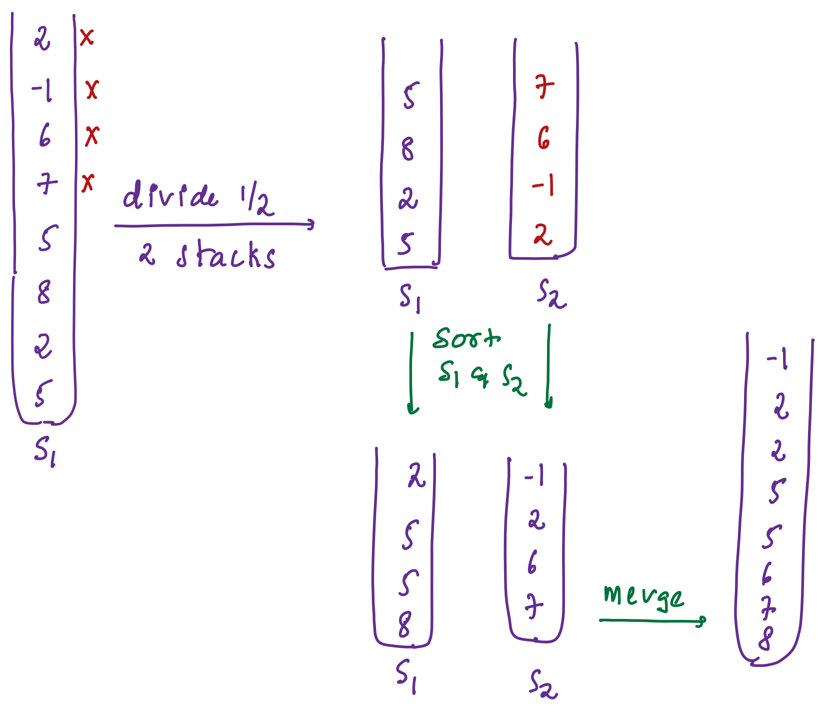
Idea: compare top of stack S_1 & S_2, smaller value is popped & is inserted into S_3 repeat it till a stack is empty
→ Copy remaining data into S_3, & finaly revern S_3



S_3 : reverse S_3

```
Stack<int> mergeStack(Stack<int> S1, Stack<int> S2){
    Stack<int> S3
    while(S1.size()>0 && S2.size()>0){
        if( S1.top() < S2.top()){
            int ele = S1.top(); S1.pop()
            S3.push(ele)
        }
        else{
            int ele = S2.top(); S2.pop()
            S3.push(ele)
        }
    }
    while(S1.size()>0) { int ele = S1.top(); S1.pop() S3.push(ele)}
    while(S2.size()>0) { int ele = S2.top(); S2.pop() S3.push(ele)}
    //revern Stack S3
    Stack<int> S4
    while(S3.size()>0) { int ele = S3.top(); S3.pop() S4.push(ele)}
    return S4
}
```

3Q) Sort Stack only using stacks



Ass: Given a stack it will sort & return it

```
Stack<int> SortStack( Stack<int> S1)   TC: O(N log N)   SC: O(log N)
                                                           ↓
                                                      recusive stack size
{
    if( S1.size() <= 1) {return S1}
    Stack<int> S2 ;
    int n = S1.size()
    i=0 ; i< n/2 ; i++) {
        ele = S1.top()
        S1.pop() // delete top
        S2.push(ele)
    }
    S1 = SortStack(S1) // it will sort & return stack
    S2 = Sort Stack(S2) // it will sort & return stack
    return mergeStack(S1, S2) // merged 2 sorted stacks & return
                                 a 1 sorted stack
}
```

Expression Evaluation: → BODMAS

Ex1: $\underbrace{8 * 5} + 4 = 44$

Ex2: $10 + \underbrace{3 * 4} - 6/3 = 20$

→ $10 + 12 - \underbrace{6/3}$

→ $10 + 12 - 2 = 20$

Ex3: $\underbrace{7 \times 1} + 2 - \underbrace{8 * 3} + \underbrace{10/5} = -13$

→ $7 + 2 - 24 + 2$

→ -13

higher

$\boxed{/ \; *}$: same predence

$+ \; -$: same predence

In above case, if 2 operators have same precedence, operator which comes first we do that

Infix: operator between operands

postfix: operator after operands

prefix: operator before operands

Infix Expressions ⟶ Post fix Expression

a + b	⟶	a b +
a − b	⟶	a b −
b − a	⟶	b a −
a * b	⟶	a b *
a / b	⟶	a b /

Infix:

: $4 + \underbrace{8 * 7}$

: $4 + \underbrace{8 7 *}$

: $4 8 7 * +$

: $10 + \underbrace{3 * 4} - 7$

: $10 + \underbrace{3 4 *} - 7$

: $\underbrace{10 3 4 * +} - 7$

: $\underline{10 3 4 * + 7 -}$

: $10 / \underbrace{(4 - 2)} * 6 + 9$

: $10 / \underbrace{4 2 -} * 6 + 9$

: $\underbrace{10 4 2 - /} * 6 + 9$

: $\underbrace{10 4 2 - / 6 *} + 9$

: $10 4 2 - / 6 * 9 +$

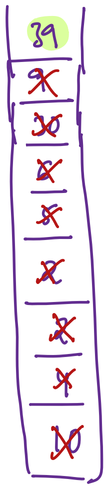
Infix: $\longrightarrow$ Postfix:

$10 / (\underset{\sim}{4-2})*6 + 9$: $10\ 4\ 2\ -\ /\ 6\ *\ 9\ +$

: $\underset{\sim}{10/2}*6+9$

: $5*6+9$

: 39



1)

a	b	a - b	
4	2	4 - 2	push

2)

a	b	a / b	
10	2	5	push

3)

a	b	a * b	
5	6	30	push

4)

a	b	a + b	
30	9	39	push

Infix Expression $\xrightarrow[\text{TODO}]{O(N)}$ Postfix Expression $\xrightarrow{O(N)}$ Evaluate Expression

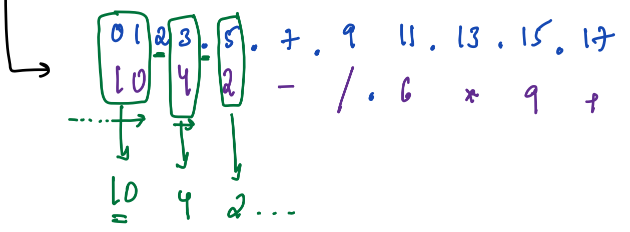
Evaluating expression : TC : O(N) SC : O(N)

→ Iterate on expression
  → If we get operand : push inside stack
  → If we get operator :
    → pop & get top 2 elements, calculate & push final result in stack
    → Note: 1st top ele is b ; 2nd top ele is a] Calculate a $\oplus$ b
      ↳ any operator

→ Once entire expression is evaluated, one element left out in stack is your final ans.

Input Postfix String:

: Between any 2 operator/operands a single space is used acts as a separator


0 1 2 3 5 7 9 11 13 15 17
10 4 2 - / 6 * 9 +
10 4 2 . . .