Stacks 1

Abhishek Sharma
Akansh Nirmal
amit khandelwal
Bhaveshkumar
Burhan
Gagan Kumar S
Gowtham
Madhan Kumar M S
Murali krishna Talluri
Naval Oli
Pankaj Bhanu
Prajwal Khobragade
Rajat Sharma
Rajendra
Sanket Agarwal
Sanket Giri
Saurabh Ruikar
sharath r
Shradha Srivastava
Shrikanth
Sneha L
Subhashini
Subhranil Kundu
Sumit Adwani
Suyash Gupta
Venkata Sribhavana Nandiraju
Vimal Kumar
Vishal Mosa
Yugesh v

AGENDA:

- → Introduction
- → Stack implementation
- → Balanced parentheis
- → Double Character Trouble
- → Evaluate postfix Exprension.

Current PSP

↓

64% → 70%

Real Life Examples

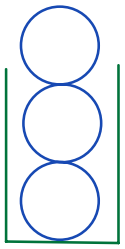
1> Glass of water

2> Pile of plates

LIFO

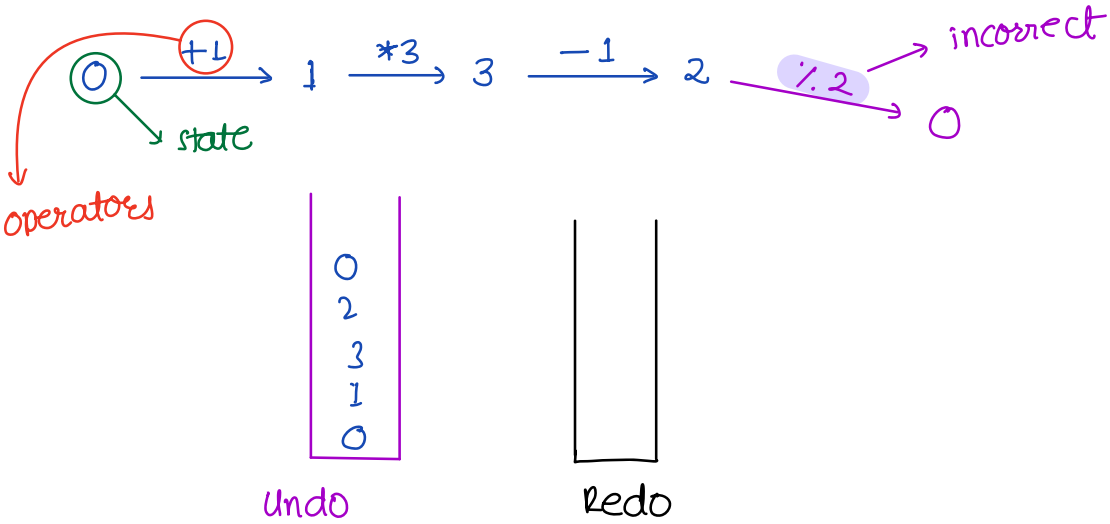
Last In
First Out

3> Box of balls



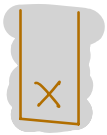
What is stack ?

Stack is a linear DS which supports LIFO



Operations of Stack

Any stack implementation supports below 4 operation



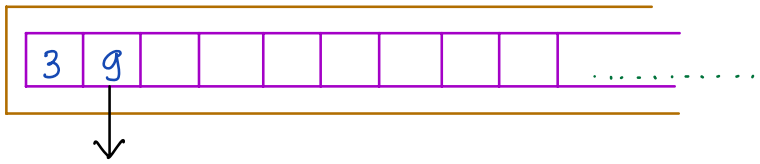
- push (x) ⟶ add value x to top of stack
- pop ( ) ⟶ removes top value from stack
- peek ( ) / top ( ) ⟶ fetch topmost value from stack
- isEmpty ( ) ⟶ if stack is empty

TC: O(1) all the above 4 operations.

push (6) ✓
isEmpty ( ) ⟶ False
pop ( ) ⟶ 6
push (7) ✓
peek ( ) ⟶ 7

Implement stack using arrays

Implement push pop peek isEmpty in TC : O(1)



t { index of top value }

intial value of t = -1

stack → [0 ..... t]

push(3)

push(4)

pop()

push(9)

peek()

isEmpty()

t--

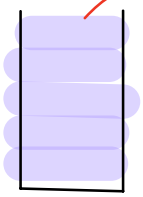
A[t]

t == -1

---

t = -1 A[]

TC : O(1)



overflow

① use dynamic array

② limit the capacity of stack

```
void push(x) {
    more cond"
    t++
    A[t] = x
}
```

```
int pop() {
    if (isEmpty()) return -1
    x = A[t]
    t--
    return x
}
```

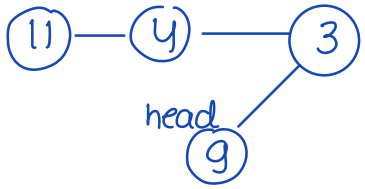
underflow

```
int peek() {
    if (isEmpty()) return -1
    return A[t]
}
```

```
boolean isEmpty() {
    return t == -1
}
```

Implement Stacks Using Linked List.

push(3) ✓
push(4) ✓
push(11) ✓
pop() ✓
pop() ✓
push(9) ✓
peek() ✓
isEmpty() ✓



Note → do not insert at tail

TC: O(1)

- push(x) → insert at head
- pop() → delete head
- peek() / top() → return head.data
- isEmpty() → head == null

Salesforce
***

Q> Check whether the given sequence of parenthesis is valid or not.

( )
{ }
[ ]

1> closing bracket ⟶ the last opening bracket should match

2> opening bracket ⟶ there should be a closing bracket.

( ) ⟶ valid

( )) (⟶ invalid

(( )) ( ) ⟶ valid

```
boolean roundBalancedParenthesis ( String s ) {
    open = 0
    close = 0
    for i ⟶ 0 to N-1 {
        if ( s[i] == '(' ) {
            open++
        } else {
            close++
        }
        if ( close > open ) return false
    }

    return open == close
}
```

TC: O(N)

SC: O(1)

→ ( ) [( )] { } → valid

→ ( ) [( )]] → invalid

▷ closing bracket → the last opening bracket should match

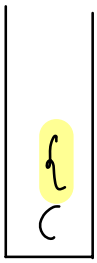
→ ({) } → invalid rule 1

→ ( ) [{ } ( )] → valid

∵ we want to keep track of last opening bracket

→ LIFO

( ) [( )] { }



({) }

↓

false

Hw. why can't we use 6 variables to solve the above?

Break : 22:38

Double Charactor Trouble

Given a string s, remove equal pair of consecutive characters multiple times till possible and return the final string

a b b c ⟶ ac

a b c c b d e ⟶ a b b d e

a d e

a b b c b b c a c x

a c c a c x

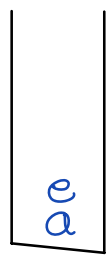
a a c x

c x

∵ we want to match latest char equal to current char on left ⟶ LIFO

a b b e

↑ ↑ ↑ ↑



ea

reverse

a b c d d c a a b x

x
~~a~~
~~d~~
~~c~~
~~b~~
a

reverse (x a)

d d a d a c

c
a
d
a
~~d~~

reverse (c a d a)

Pseudo code

```
st   // init stack

for i ---> 0 to N-1 {
    char = s[i]
    if ( !st.isEmpty() && char == st.peek() ) {
        st.pop()
    }
    else {
        st.push(char)
```

// Learn how to optimise this using StringBuilder

```
String ans = "";
while (!st.isEmpty()) {
    ans += st.pop()
}
// reverse ans
return ans
```

TLE
$\because$ TC : $O(N^2)$

Infix Expressions

Postfix Expression

$(a + b) \longrightarrow a\ b\ +$

$a * (b - c) \longrightarrow a\ b\ c\ -\ *$

$(a * b) - c \longrightarrow a\ b\ *\ c\ -$

operand operator operand

operand operand operator

$\underbrace{2\ 3\ +}_{2+3\ =\ 5}$

$4\ \underbrace{3\ 3\ *}\ +\ 2\ -$

$\underbrace{4\ 9\ +}\ 2\ -$

$13\ 2\ -$

$13 - 2 = 11$

Evaluate the given valid postfix expression

$$\underbrace{10 \quad 6 \quad -}_{10 - 6 \;=\; 4}$$

$$\underbrace{5 \quad 2 \quad *}_{10} \quad \underbrace{3 \quad -}_{7}$$

$$\underbrace{3 \quad 5 \quad +}_{8} \quad 2 \quad - \quad 2 \quad 5 \quad * \quad -$$

$$\underbrace{8 \quad 2 \quad -}_{6} \quad 2 \quad 5 \quad * \quad -$$

$$6 \quad \underbrace{2 \quad 5 \quad *}_{10} \quad -$$

$$\underbrace{6 \quad 10 \quad -}_{6 - 10 \;=\; -4}$$

$$\underbrace{6 - 10 = -4}$$

-4	3 5 + 2 - 2 5 * -
~~10~~	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
~~5~~	
~~2~~	
~~6~~	
~~2~~	
~~8~~	
~~5~~	
~~3~~	

Pseudocode

```
int evaluate( String[] A ) {
    st // init stack

    for ( i ---> 0 to N-1 ) {
        char = A[i]
        if ( char is an operator ) {
            x = st.pop()
            y = st.pop()
            z = y op x
            st.push(z)
        }
        else {
            st.push( int(char) )
        }
    }
    return st.pop()
}
```

Be careful

TC: O(N)

SC: O(N)

Problem solving Framework

Q

solve < 25 → All Good

! solve < 25 →

1. Hints
2. Video sol^n
3. Raise TA
4. Post in WA grp
5. Reach out to me

push (3) ✓
push (11) ✓
push (7)
pop ( )
pop( )
push (15)

Head null

3 → null

11 → 3 → null

7 → 11 → 3 → null

s = "abacdef" N

s = s + 'z' TC: O(N)