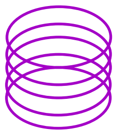
Stacks 1

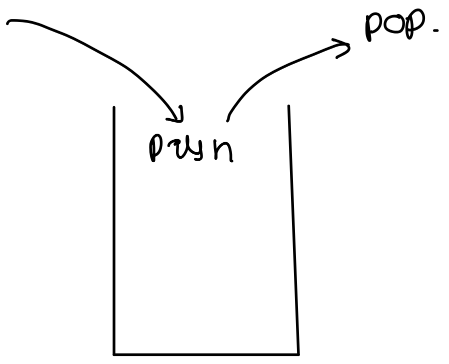
→ Introduction
→ Stack implementation
→ Balanced parenthesis
→ Double character Trouble
→ Evaluate postfix Expresion.

Real Life Examples

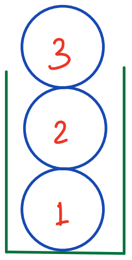
1> Glass of water

2> Pile of plates





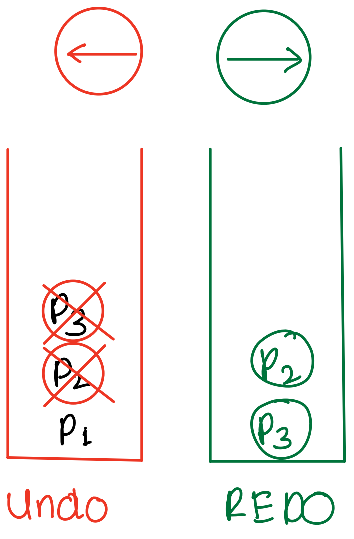
3> Box of balls

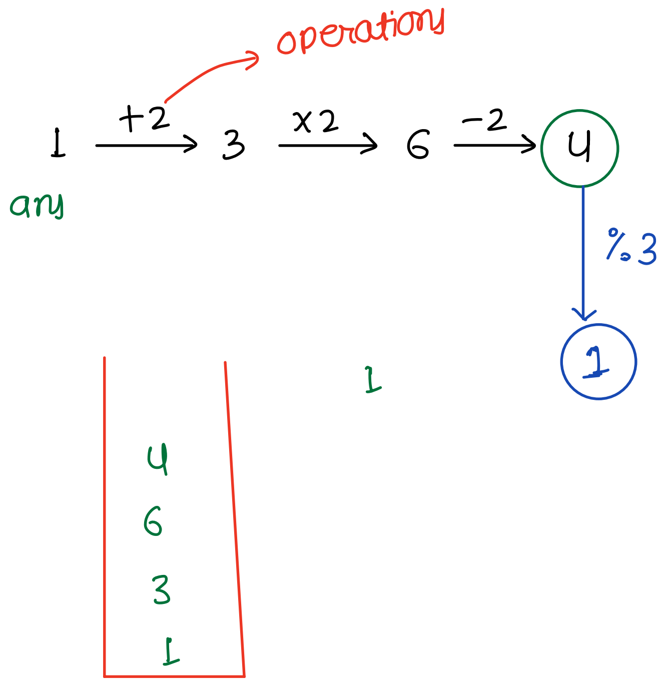


Last in first out

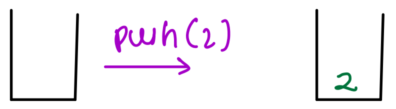
what is stack ?

So stack is a LIFO data structure that supports push , pop , peek or top , isEmpty operation on it.

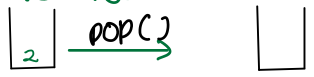




Operations of Stack



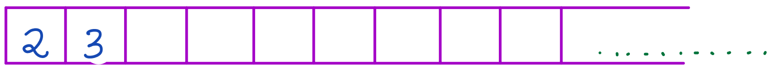
- push (x) ⟶ Push the value x on top of stack



- pop ( ) ⟶ Remove the value on top of stack
- peek() / top ( ) ⟶ Get value stored on top of stack.
- isEmpty () ⟶ Check if stack is empty or not.

All of the above operations take O(1) time

Implement stack using arrays



$\longrightarrow$ push(2) ✓

$\longrightarrow$ push(3) ✓

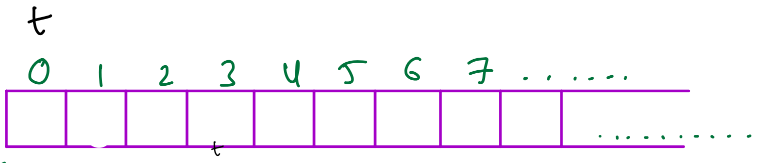
$\longrightarrow$ peek() ✓ $\longrightarrow$ 3

$\longrightarrow$ pop() ✓

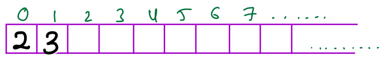
$\longrightarrow$ peek() $\longrightarrow$ 2

stack is represented by index 0 to t

If the stack is empty t = −1



$\longrightarrow$ push(2) ✓ t=0



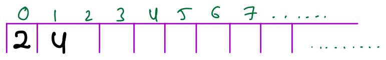
$\longrightarrow$ push(3) ✓ t=1

$\longrightarrow$ peek() ✓ $\longrightarrow$ A[t] $\longrightarrow$ A[1] = 3

$\longrightarrow$ pop() ✓ $\longrightarrow$ decrement t. t=0

$\longrightarrow$ peek() $\longrightarrow$ A[0] $\longrightarrow$ 2

$\longrightarrow$ push(4) $\longrightarrow$ t=1 A[t] = 4

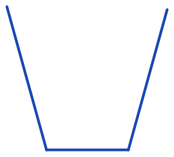


```
stack [] , t = -1

void push ( x ) {
        t += 1
        A[t] = x
}
```

overflow $\downarrow$ dynamic arrays.

```
void pop ( ) {
    if (!isEmpty ())
        t -= 1
}
```



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

```
boolean isEmpty ( ) {
        if (t == -1)
                return true
        else
                return false
}
```

$\left.\begin{array}{l}\text{if (t == -1) return true}\\ \text{else return false}\end{array}\right\}$ return t == -1

All of the above operations take O(1) time

Implement stacks using Linked List.

push (2) $\longrightarrow$ 1 $\longrightarrow$ null

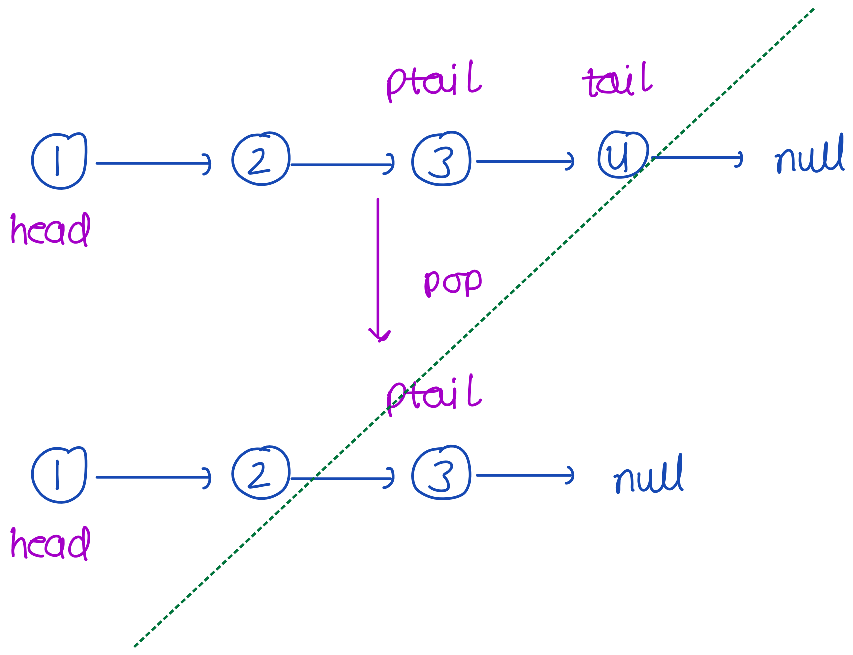
pop ( ) $\longrightarrow$

push (3) $\longrightarrow$ 3 $\longrightarrow$ null

push (4) $\longrightarrow$

peek()

pop ( )



$\longrightarrow$ push and pop from head

push (1) 1 $\longrightarrow$ null

push (2) 2 $\longrightarrow$ 1 $\longrightarrow$ null

push (3) 3 $\longrightarrow$ 2 $\longrightarrow$ 1 $\longrightarrow$ null

pop ( ) 2 $\longrightarrow$ 1 $\longrightarrow$ null

peek $\longrightarrow$ head.val isEmpty head == null

All of the above operations take O(1) time

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

1> closing bracket $\longrightarrow$ the last opening bracket should match

( )
{ }
[ ]

2> opening bracket $\longrightarrow$ there should be a closing bracket.

$\longrightarrow$	( )	balanced
$\longrightarrow$	) (	not balanced
$\longrightarrow$	(( ))	balanced
$\longrightarrow$	( )) ( ) ( )	not balanced.

Idea $\longrightarrow$ keep track of opening and closing brackets

TC: O(N)   SC: O(1)

```
boolean roundBalancedParenthesis ( String s ) {
        open = 0
        close = 0

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

$\longrightarrow$ ( ) [( )] { } $\longrightarrow$ yes

$\longrightarrow$ ( ) [( )]] $\longrightarrow$ no

$\longrightarrow$ ({) } $\longrightarrow$ no because rule 1

$\longrightarrow$ ( ) [{ } ( )] $\longrightarrow$ yes

( ) [{ } ( )]

$\uparrow$

stack

return true

({) }

$\uparrow$

{)

stack ({

return false

( ) [( )]]

$\uparrow$

TODO :

solve in assignment

stack

return false

8 : 54 am

Double Character Trouble

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

a b b c ⟶ a c

a b c c b d e

⟶ a b b d e

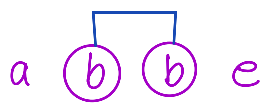
⟶ a d e

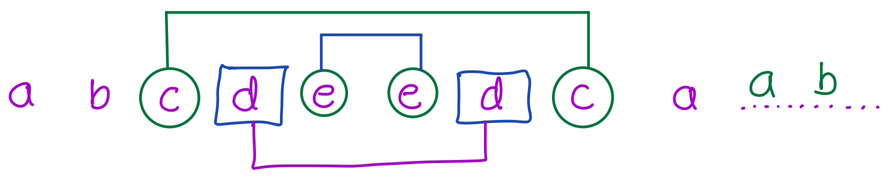
a b b c b b c a c x

⟶ a c c a c x

⟶ a a c x

⟶ c x





a b c d d c a a b x

$\uparrow$

stack a x

$\longrightarrow$ "ax"

Pseudocode

```
String doubleTrouble ( String s ) {
    stack        //  Check declaration

    for ( i → 0 to N-1) {
        Char = s[i]
        if ( stack is not empty &&
            char == stack.peek()) {
                stack.pop()
        }
        else {
            stack.push()
        }
    }

    ans = ''
    while (! stack.isEmpty) {
        ans += stack.pop()
    }
    return reverse of ans
}
```

use string Builder or equivalent in your language.

In your language pop is void or it return the element

TC : O(N)
SC : O(N)

Infix Expressions **Postfix Expression**

$a + b \longrightarrow a\ b\ +$

$a * b - c \longrightarrow \boxed{a\ b\ *}\ c\ -$

$a * (b - c) \longrightarrow a\ \boxed{b\ c\ -}\ *$

$$\begin{tikzcd}
2 * (3 - 1) \arrow[r] & 2\ \boxed{3\ 1\ -}\ * \arrow[d] \\
& \boxed{2\ 2\ *}
\end{tikzcd}$$

$\underline{\underline{4}}$

Evaluate the given valid postfix expression

10 6 − $\longrightarrow$ return 4

$\boxed{3 \quad 5 \quad +}$ 2 − 2 5 * −

$\longrightarrow$ $\boxed{8 \quad 2 \quad -}$ 2 5 * −

$\longrightarrow$ 6 $\boxed{2 \quad 5 \quad *}$ −

$\longrightarrow$ 6 10 −

$\longrightarrow$ 6 − 10 = $\underline{\underline{-4}}$

what data structure we can use to keep track of last value ?

stack

3 5 + 2 − 2 5 * −

$\uparrow$

stack = $\underline{\underline{-4}}$

```
int evaluate ( String [] s ) {
    stack = []    // in your language.

    for (i ⟶ 0 to N-1) {
        char = s[i]
        if ( char is an operator ) {
            // pop out last two values.
            val2 = stack.pop()
            val1 = stack.pop()

            // Based on result of val1, val2
            // operator push the result in stack

            res = helper( val1, val2, char)
            stack.push(res)
        }
        else {
            int num = convert (string to int)
            stack.push(num)
        }

    }
    return stack.peek()
}
```

TC : O(N)
SC : O(N)

Doubt session

3 5 + 2 - 2 5 * -

	stack
$3 \longrightarrow$	3
$5 \longrightarrow$	3 5
$+ \longrightarrow$	8
$2 \longrightarrow$	8 2
$- \longrightarrow$	6
$2 \longrightarrow$	6 2
$5 \longrightarrow$	6 2 5
$* \longrightarrow$	6 10
$- \longrightarrow$	-4