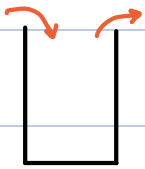
SCALER

Stacks 1

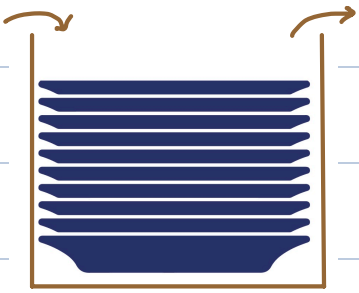
TABLE OF CONTENTS

Stacks

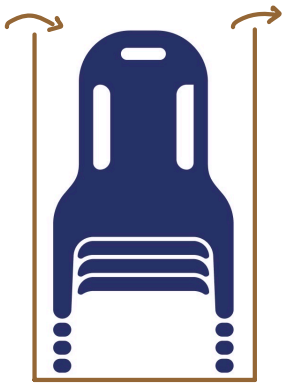


LIFO Last In First Out

1. Stack of Plates

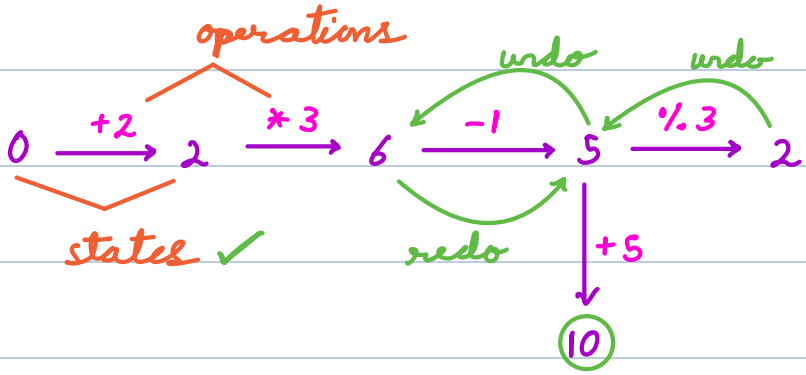


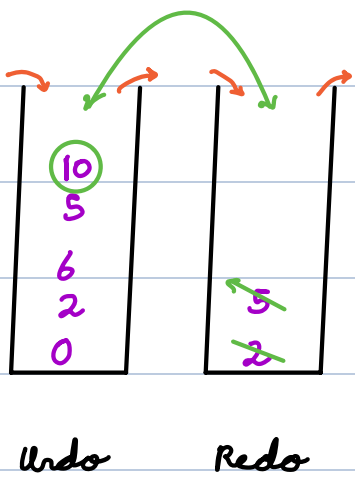
2. Stack of Chairs



3. Stack of Functions (Recursion)

4. Undo / Redo



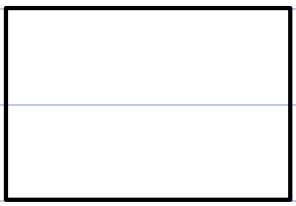


If we perform a new operation → the redo stack becomes empty.

Scenario

In a Flipkart warehouse, boxes are stacked one over another. Each box is tagged with a weight, and for safety, heavier boxes must be placed below lighter ones. Workers need a system to check if adding a new box on top is safe.

Workers want to perform two kinds of operations:-

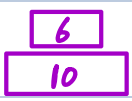


Type ADD : a new box of some weight on the top

Type REMOVE : the topmost box

Example :

Query type	Value	Answer (True/False)
ADD	10Kg	true ✓
ADD	5Kg	true ✓
ADD	12Kg	false (not allowed) ✗
REMOVE	-	true ✓
ADD	6Kg	true

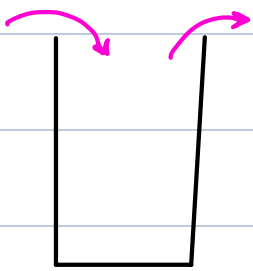


insert → if (wt of new box > wt of top box)
return false
else return true

Operations in Stack

1. **Push (x)** - Insert X on the top of stack.
2. **Pop ( )** - Remove the top element.
3. **Top ( ) / Peek ( )** - check / Get the top element.
4. **isEmpty ( )** - checks if stack is empty.

TC = O(1)

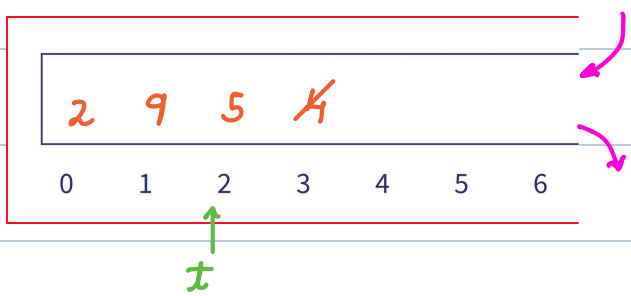


Implementing Stacks using Arrays

push (2)
push (9)
push (3)
peek () → 3
push (4)
pop () → 4
pop () → 3
push (5)

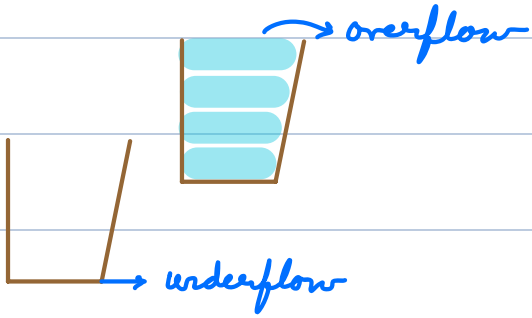
Stack

0 —— t



t = -1

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



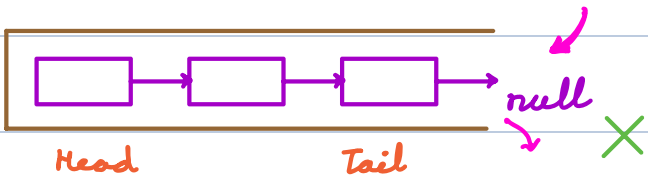
```
void push (x) {                    // overflow
    t++                            1) Use dynamic array ✓✓
    A[t] = x                       2) Do not insert if array
}                                     is full. (t == N-1)
```

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

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

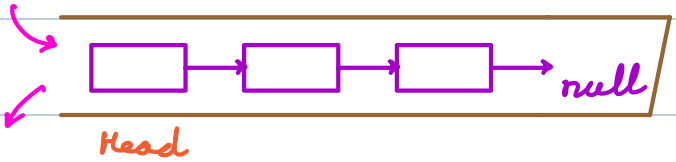
TC ∀ operations → O(1)

Implementing Stacks using Linked List



insert as last node → TC = O(N) → O(1)

removing last node → TC = O(N)



insert head → TC = O(1)

remove head → TC = O(1)

push (x) → insert x as head

peek () → return head data

pop () → remove head node

} check underflow

isEmpty () → check if head is null

TC = O (1)

< *Question* > : Check whether the given sequence of parenthesis [ ], { }, ( ) is valid or not?

Example : ()[{}()] ✓

()[] ✓

({}) ✓

({)} x

([{]) x

([{}[]()]) ✓

([{}[]()])

([{}

store data but use latest data first ⇒ LIFO ⇒ stack

open bracket → store

close bracket → check & remove pair.

([{}[]()])

~~(~~	{} ✓
~~[~~	[] ✓
~~{~~	[] ✓
~~[~~	() ✓
~~(~~	() ✓

```
// stack → st
for i → 0 to (N-1) {
    ch = s[i]
    if (ch == '{' || ch == '(' || ch == '[') {
        st.push(ch)
    } else if (ch == '}') {
        if (st.peek() != '{') return false
        st.pop()
    } else if (ch == ']') {
        if (st.peek() != '[') return false
        st.pop()
    } else {
        if (st.peek() != '(') return false
        st.pop()
    }
}
return st.isEmpty()
```

TC = O(N) SC = O(N)

Double Character Trouble

Given a string str. Remove equal pair of consecutive elements till possible.

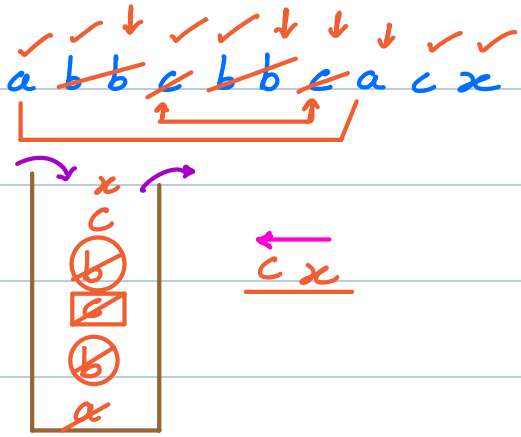
Return the strings without adjacent duplicates.

1. a ~~b b~~ d → ad
2. a b ~~c c~~ b d e → a ~~b b~~ d e → a d e
3. a ~~b b~~ b d → a b d
4. 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~~

store data but use latest data first ⇒ LIFO

⇒ stack



```
// stack → st
for i → 0 to (N-1) {
    ch = s[i]
    if (st.isEmpty() || (st.peek() != ch))
        st.push(ch)
    else    st.pop()
}

ans = ""
while (!st.isEmpty()) {
    ans = st.pop() + ans
}

return ans
```

TC = O(N) SC = O(N)

infix	postfix
2 + 3	2 3 +
2 + (5*3)	2 5 3 * +
operand1 operator operand2	operand1 operand2 operator

< *Question* > : Evaluate given valid post fix expression.

arr [ ] -

3 + 5 = 8

2 * 5 = 10

8 - 2 = 6

6 - 10 = -4 (Ans)

Quiz : 5 , 2 , * , 3 , -

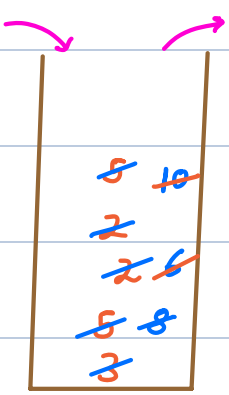
operator → select latest 2 operands & evaluate.

5 * 2 = 10

10 - 3 = 7 (Ans)

✓ ✓ ↓ ✓ ↓ ✓ ✓ ↓ ↓

3 5 + 2 - 2 5 * -



y = st.pop()

x = st.pop()

x operator y

3 + 5 = 8

8 - 2 = 6

2 * 5 = 10

6 - 10 = -4 (Ans)

H.W → code using stack.

TC = O(N) SC = O(N)