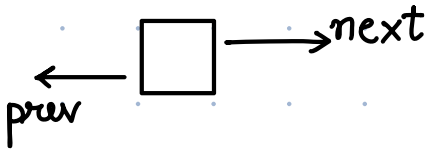
Revision —

1. Easier traversal in both directions
2. 

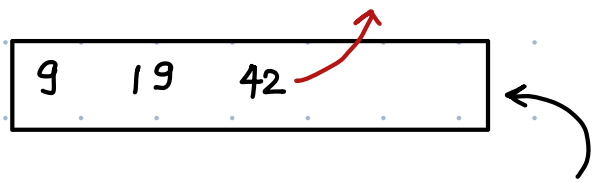
Stacks

Arrays

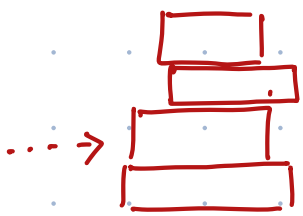
LL

Stacks - LIFO It follows LIFO principle

Last in First Out



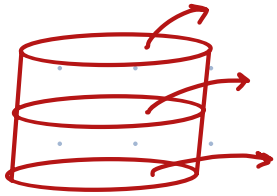
Pile of Books



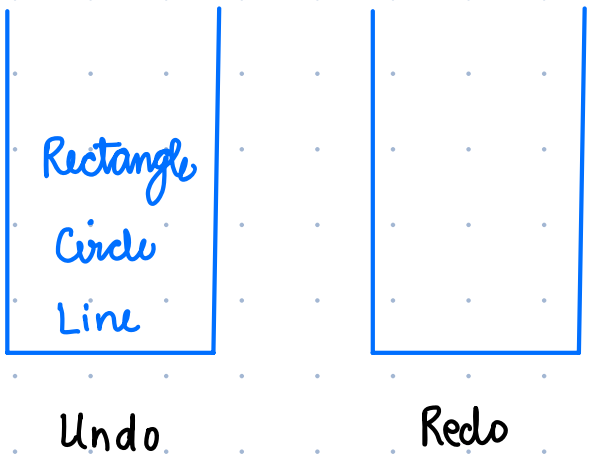
Stack of plates

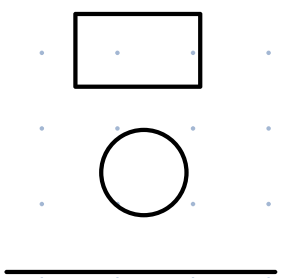
Stack of chairs

Idly stand

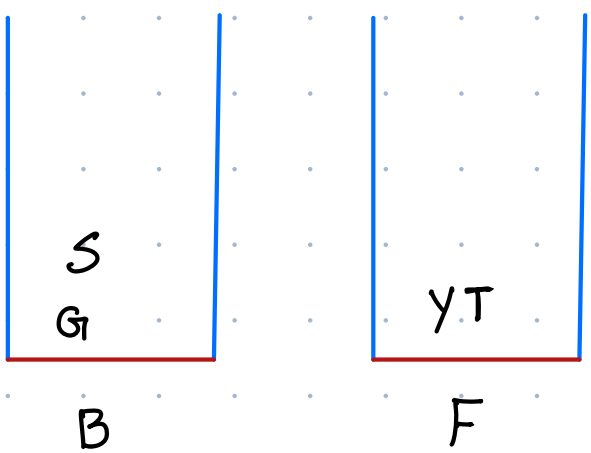


Undo/redo button





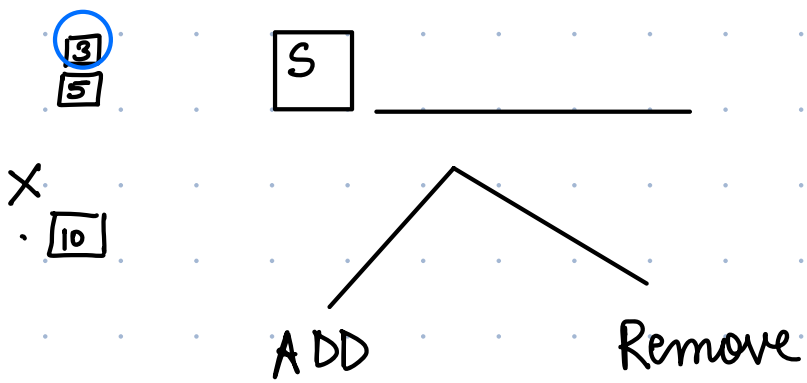
Browser



Recursion

Evaluation of postfix expressions

FlipKart Warehouse Management

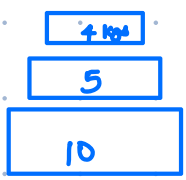


ADD 5 kg

Remove 10 kg

ADD 3 kg

Checking topmost element.



10 kg ADD

5 kg ADD

12 kg Remove

4 kg ADD

Steps:

1. To find topmost box ---> access top of stack

2. New box | Topmost box

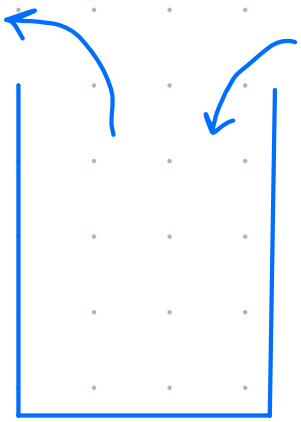
NB > TB : Remove

NB < TB : ADD

---

2 things:

1. Focus on topmost element
2. Both operations happen from same side.



My own stack →

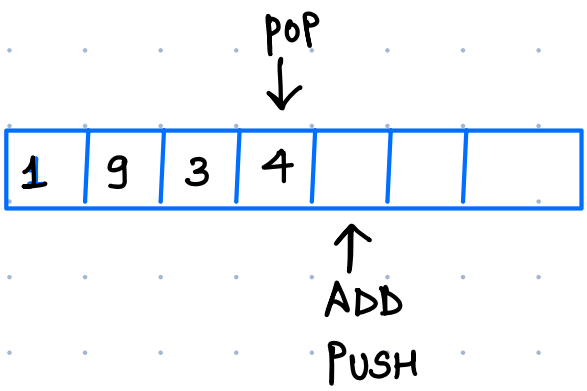
push (x) : ADD

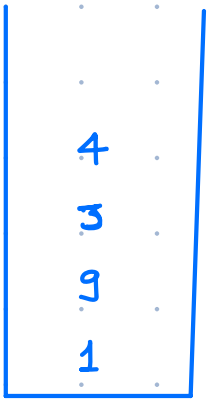
pop ( ) : Remove

puk ( ) : access the topmost element

isEmpty ( )

Array as Stack

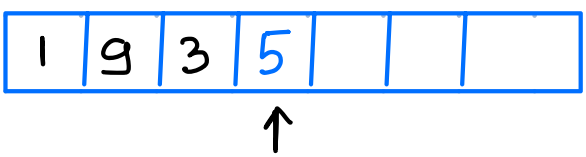




Push (1)
Push (9)
Push (3)
Push (4)
Pop()

A

Push (5)



Push (1)
Push (9)
Push (3)
Push (4)

3
9
1

top = -1

```
Push ( x ) {
    if (top == -1) { "STACK EMPTY" }
    top++ i
    A[top] = x ;

}
```

```
int Peek ( ) {
    return A[top]
}
```

```
void pop () {
    if (top. --1) { "STACK EMPTY", return }
    top--;
}
```

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

ArrayList

Empty Stack → top = -1

pop

↳ Underflow

Overflow :

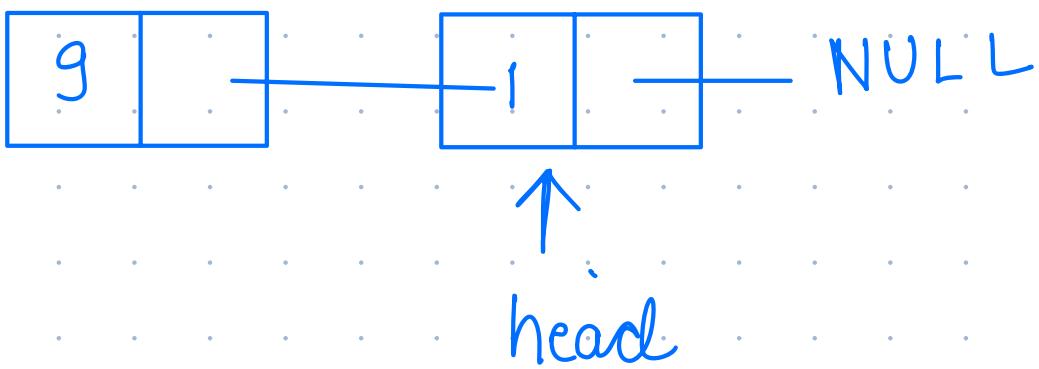
Fixed size

top == N-1 N

0 to N-1

```
if (top >= N-1) {
    Print ("Stack is full")
    return;
}
```

1 9



push

peek .... head

pop head = head.next

isEmpty : head == NULL

true

head = null

function push (data) {

nn = Create a node with data

nn.next = head

head = nn

}

function peek () {

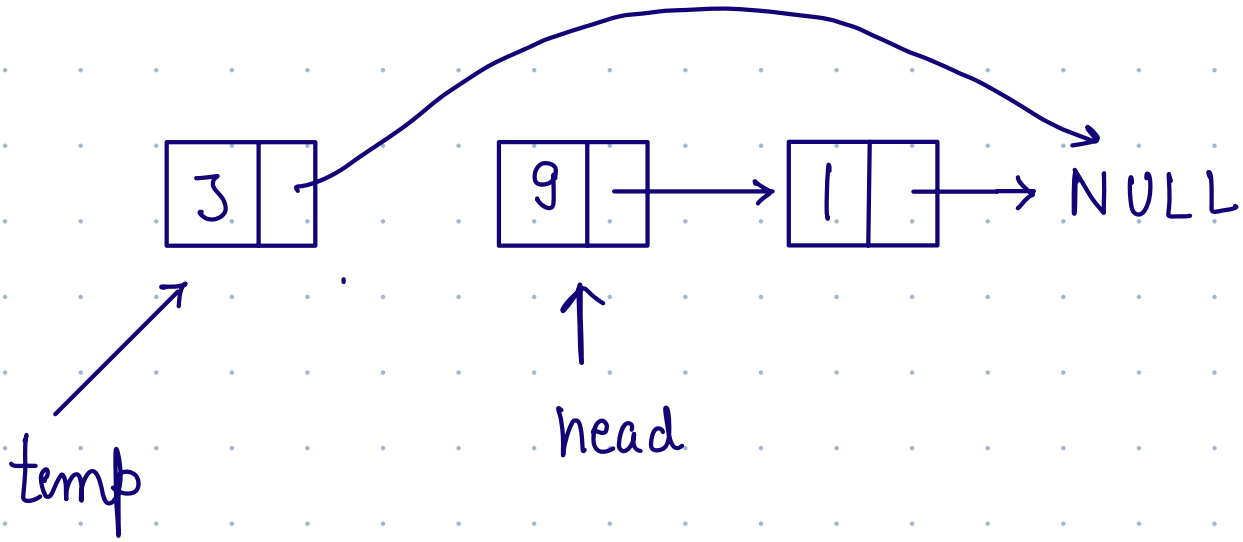
return head.data.

}

```
function isEmpty() {
    return head == null;
}

function pop() {
    if ( isEmpty() == true) { return }
    temp = head
    head = head.next
    temp.next = null
}
```

Underflow → Stack is Empty


3
9
1
NULL
temp
head

Q. { }

[ ]

( )

sequence of brackets → whether it is valid?

(({ }))

every opening bracket has a matching closing bracket

Valid?

{ { })

False

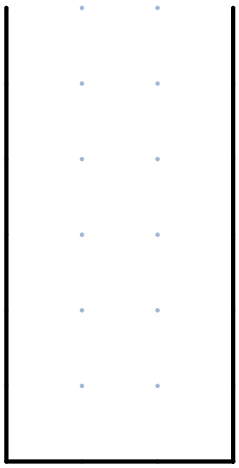
{ { { } }

no of op brackets > closing brackets

False

{ [[ ] { }] } ( ) ( ) { {

CE



if (stack is empty) {
    return false
}

Pick the topmost element from stack

if (topmost ⟷ CE) {
    return false
}

if stack is not empty ⟶ return false

else

return true

```
function validbrackets (String seq ) {
    // take an empty stack
    (for each char in seq) {
        if (opening bracket) { Put in stack }
        else {
            if (stack is empty) {
                return false
            }
            Pick the topmost element from stack
            if (topmost <--> CE) {
                return false
            }
        }
    }
}
```

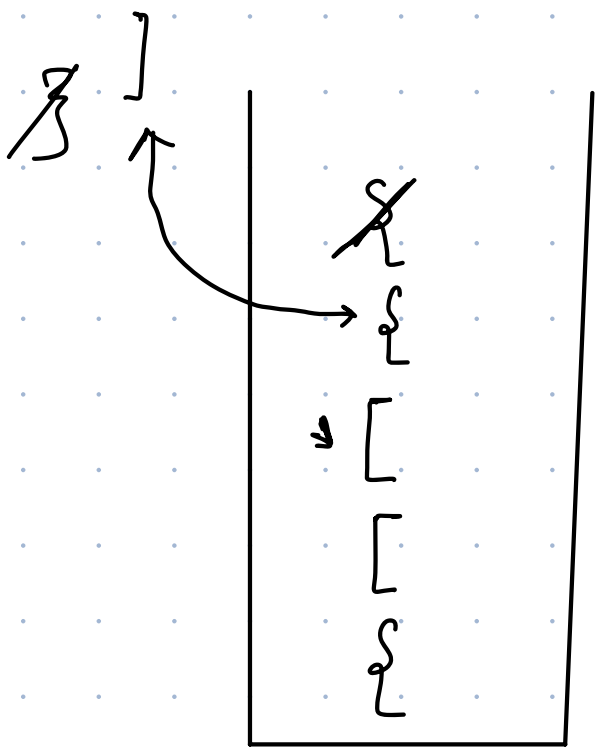
if stack is not empty ⟶ return false

else

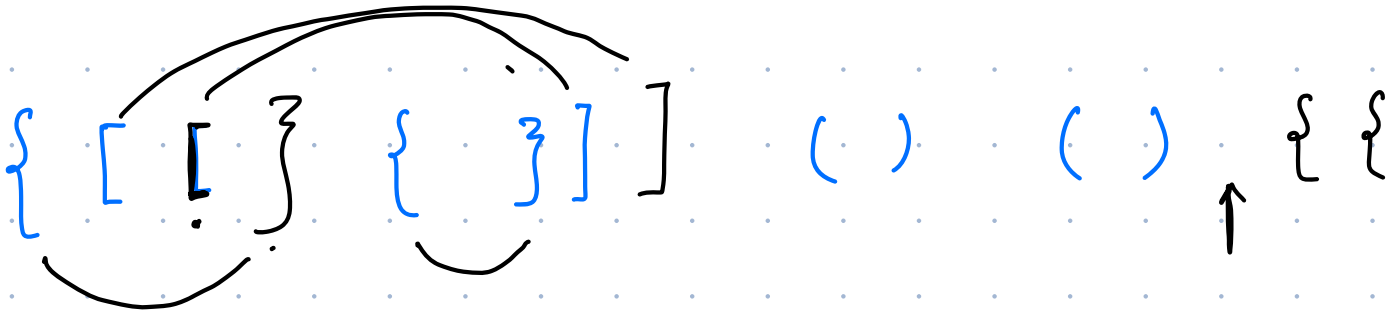
return true

}

{ [[{ { }] } ( ) ( )

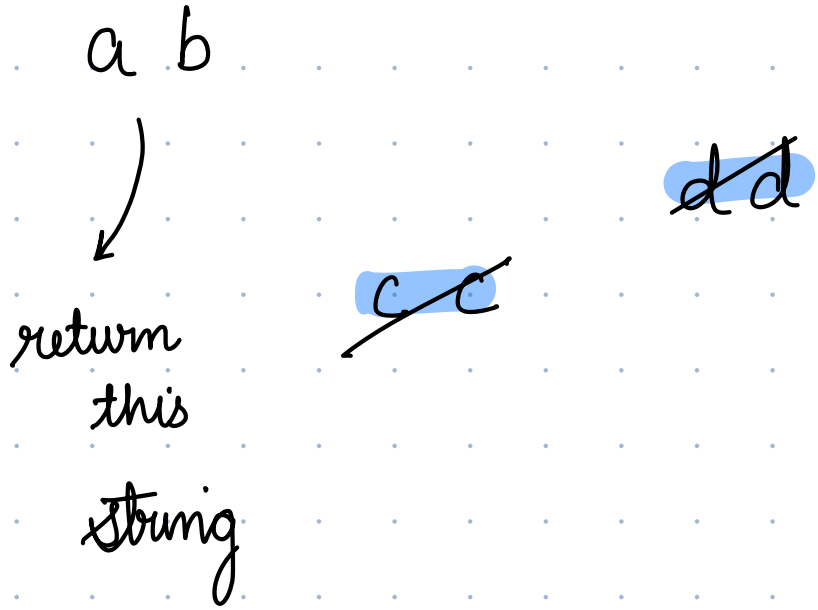


false



opened at last → closed first

Q4 Remove equal pair adjacent elements.



d d

c c

b b

a g f a

h h

c c d d

---

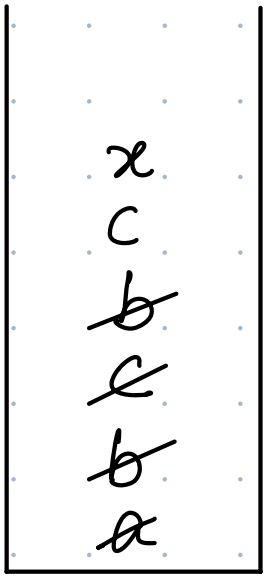
c x

b b a a c c b b

a b b c b b c a c x

CE

TE



func remove-equal-pairs (String seq) {

Initialize an empty stack

for each ch in stack {

if (st is not empty) {

TE ⟷ CE

if (match) { remove TE }

else {

add CE to stack

}

```
    } else {
        add CE to stack
    }
}
String res = "";
while (st is not empty) {
    Pop an element => ele
    res = ele + res
}
return res
}
```

Q. Evaluate Postfix expression

2 3 | * |

operands

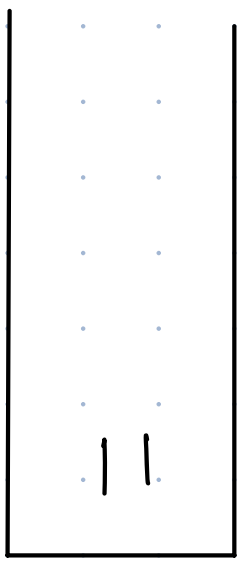
= 2 x 3 = 6

4 3 3 * + 2 −

2
13 −

3 *
3

9 +
4



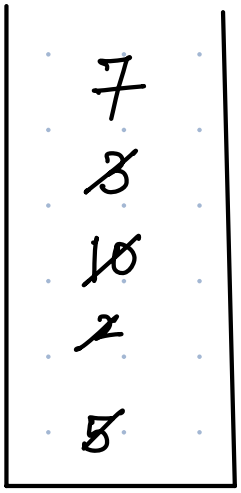
11 ans

5 2 * 3 −

10

10 3 −

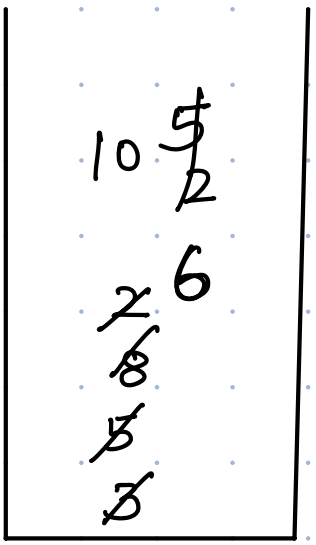
10 − 3 = 7



3 5 + 2 − 2 5 * −

6 − 10 = −4

ans



```
function evaluatePost(string exp) {
    Init an empty stack
    for each every char {
        if element is operand {
            Put it in stack
        } else {
            Remove op1
            Remove op2
            Apply operation
            Put res back into stack
        }
    }

    Pop and return the result.
}
```