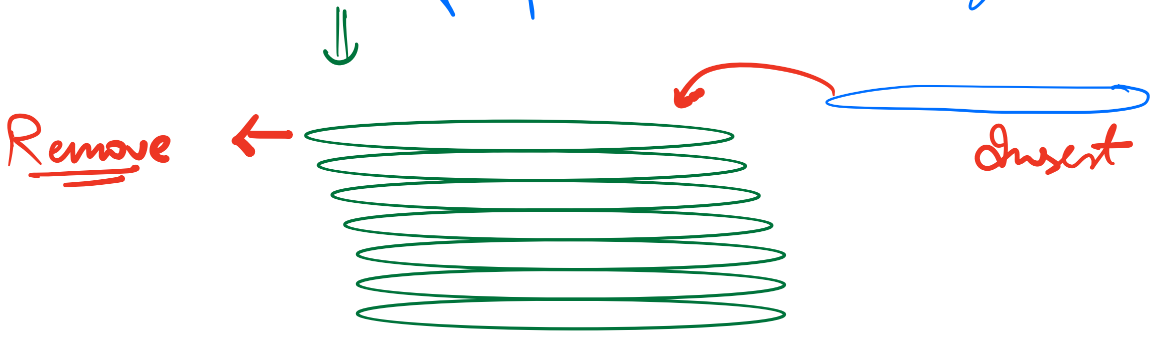
Contest ⇒ Wednesday 24th (

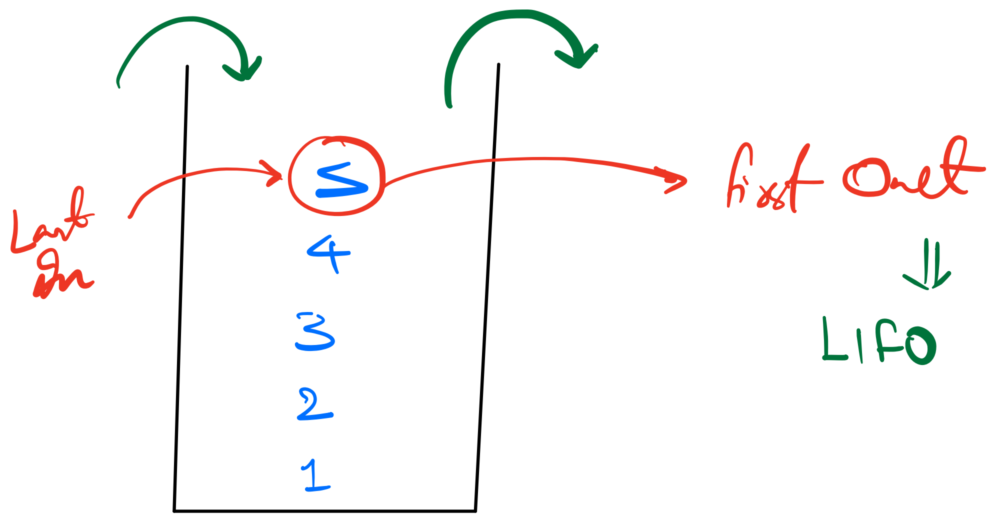
26th January ⇒ Off

Saturday ⇒ 9:00PM (Extra Class)
⇓
Queues.

Stacks

1) Stack (Pile of plates in a Buffet)

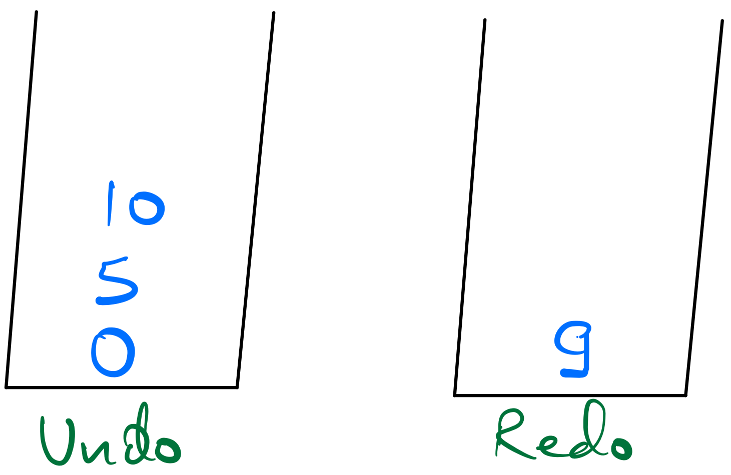




Use Cases

1) To store recursive function calls.

2) Undo / Redo



Operations

1) push (data)

↳ Insert the data at the top of stack.

2) pop()

↳ Removes & returns the top element of stack.

3) peek() / top()

↳ Return the top element of stack

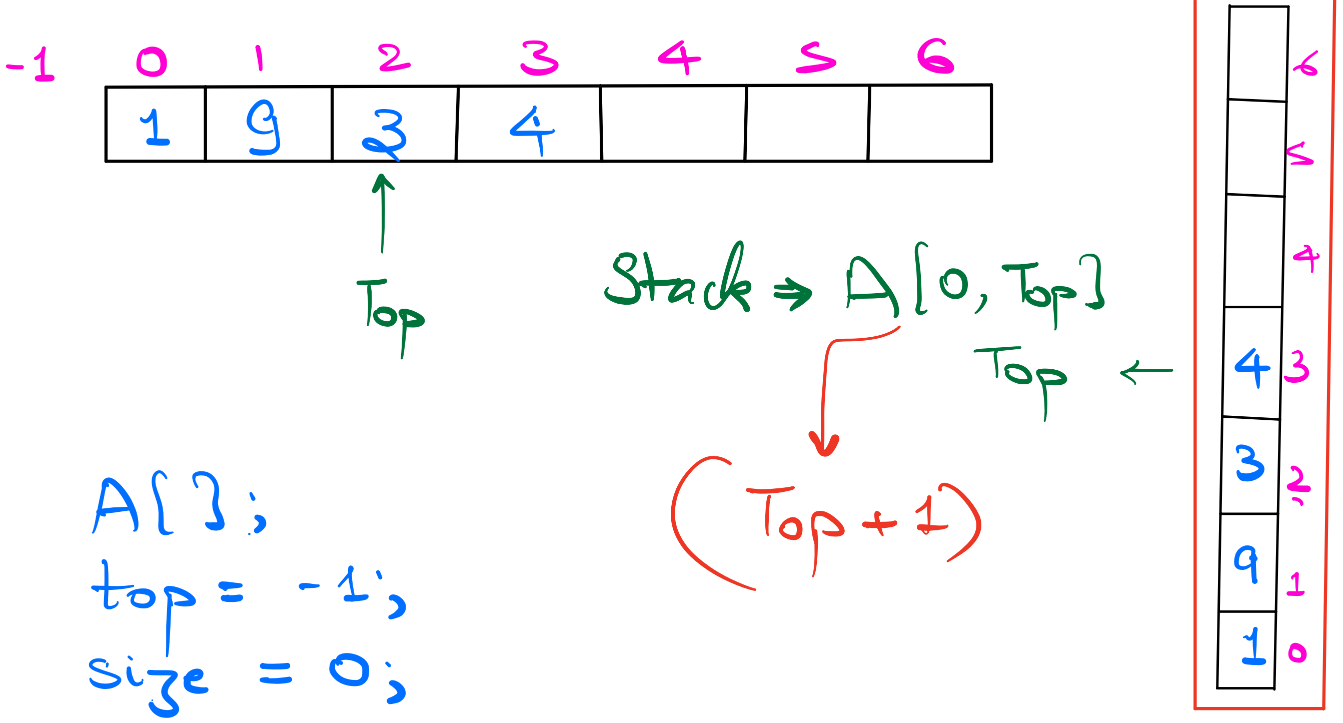
4) size()
↳ Return no. of elements

5) isEmpty()
↳ Return true if stack is empty.
false otherwise.

T.C. = O(1)

Implementation

1) Array Implementation



A[ ];
top = -1;
size = 0;

```
void push (int x) {
        top++;
        A[top] = x;
        size ++;
}


int peek() {
    if (top == -1) { return INT_MIN; }

    return A[top];
}

int pop () {

    if (top == -1) { return INT_MIN;

    int x = A[top];
    top--;
    size --;
    return x;
}

int size() {

    return size; (return top+1;)
}
```

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

$T.C. = O(1)$

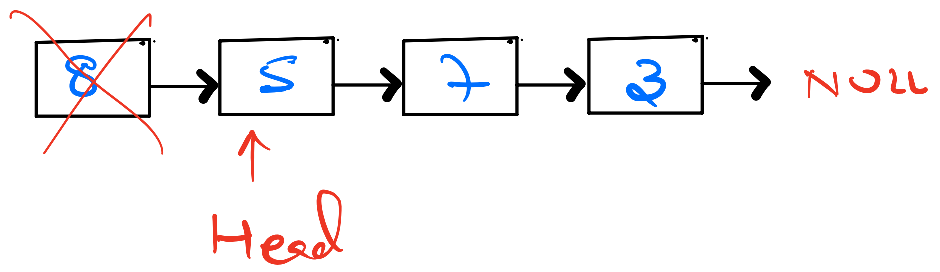
Overflow

```
void push (int x) {
        top++;
        if (top > A.size()) {
            return;
        }
        A[top] = x;
        size++;
}
```

Underflow

Linked List

I/P ⇒ ~~3~~, ~~7~~, ~~5~~, ~~8~~, peek(), pop(), pop(), peek()



```
int peek(){
    if (Head == NULL) {
            return INT_MIN;
    }
    return Head.data;
}
```

$T.C. = \underline{O(1)}$

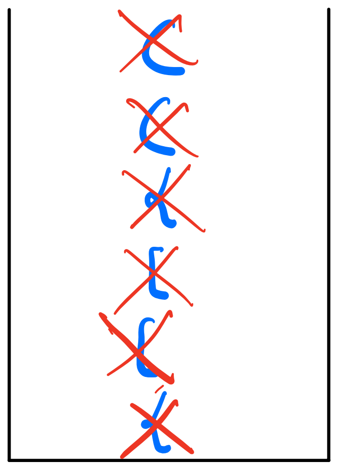
Q Check whether a given sequence of parenthesis is valid.

{ } ⇒ Curly
[ ] ⇒ Square
( ) ⇒ Round.

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

$\{ 3 \times [2 + \overset{12}{\underline{\cancel{(}5+7\cancel{)}}} \times \overset{8}{\underline{\cancel{(}3+5\cancel{)}}}] \}$

Eg → { [[ ] { }] } ( ) ( )



Code

```
bool isValid (string s) {

    HashMap <char, char> mapping;
    Stack <char> st;
    mapping.insert (')', '(');
    mapping.insert ('}', '{');
    mapping.insert (']', '[');
```

```
for (i=0; i<s.length; i++){
        if (s[i] == '(' || s[i] == '{' ||
            s[i] == '[' ) {
              st.push(s[i]);
        }
        else {
          if (st.isEmpty() || st.top() != 
                                mapping.get(s[i]) ){
              return false;
          } else {
              st.pop();
          }
        }
    }
    if (st.isEmpty()) {
          return true;
    } else {
        return false;
    }
}
```

$T.C. = O(N)$

$S.C. = \underline{O(N)}$

Q. Given a string. Remove equal pair of consecutive characters.

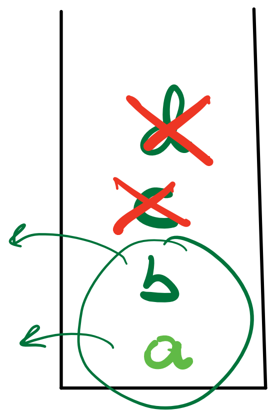
$S = a\,\underline{b\,b}\,c \Rightarrow ac$

$S = a\,b\,c\,\underline{d\,d}\,c \Rightarrow a\,b\,\underline{c\,c} \Downarrow ab$

Sol^n

```
st.push (A[0]);
∀ i>0 ⇒  if (A[i] == st.peek()) {
                st.pop();
         }
         else {
                st.push (A[i]);
         }
```

$a\,b\,c\,\underline{d\,d}\,c$



T.C. = O(N)
S.C. = O(N)

$a\underline{bb}cbbcacx$

$\downarrow$

$ac\underline{bb}cacx$

$\downarrow$

$a\underline{cc}acx$

$\uparrow$

$\underline{aa}cx$

$\downarrow$

$\boxed{\underline{\underline{cx}}}$

$b\underline{aa}a \Rightarrow \boxed{\underline{\underline{ba}}}$

Post fix Expressions

$\underset{\uparrow \atop \text{Operand}}{2} \quad \overset{\text{Operator} \atop \uparrow}{+} \quad \underset{\uparrow \atop \text{Operand}}{3} \Rightarrow$ Infix Expression

Post fix $\Rightarrow$ Operand 1, Operand 2, Operator

$\Downarrow$

$2 \quad 3 \quad +$

Conversion from Infix to Postfix

$2 + 3 - 6 \times 5 \Rightarrow (2 \; 3 +) - 6 \times 5$

$\Downarrow$

$(2 \; 3 +) - (6 \; 5 \times)$

$\Downarrow$

Single digit no's

$\underline{2} \; \underline{3} + 6 \; 5 \times -$

$\Downarrow$

seperator

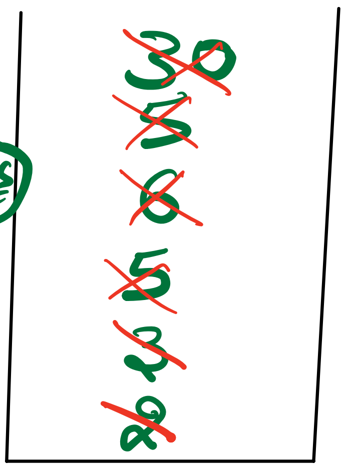
Evaluate Post fix Expression

$2 \; 3 + 6 \; 5 \; \times \; -$

stack $\Downarrow$ $\rightarrow$

$5 - 30 = \boxed{-25}$

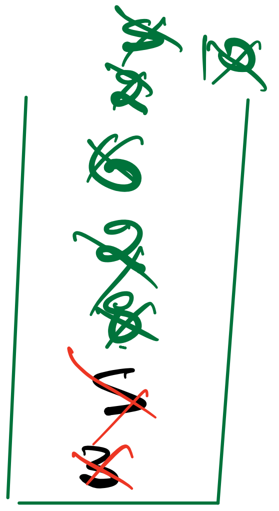
$2+3=5 \quad 6 \times 5 = 30$



$T.C. = O(N)$

$S.C. = O(N) \Rightarrow$ Stack

3 5 + 2 − 2 5 * −



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

$2 \times 5 = 10$

$8 - \cancel{2} = 6$

$\Downarrow$

70% PSP

$\Downarrow$

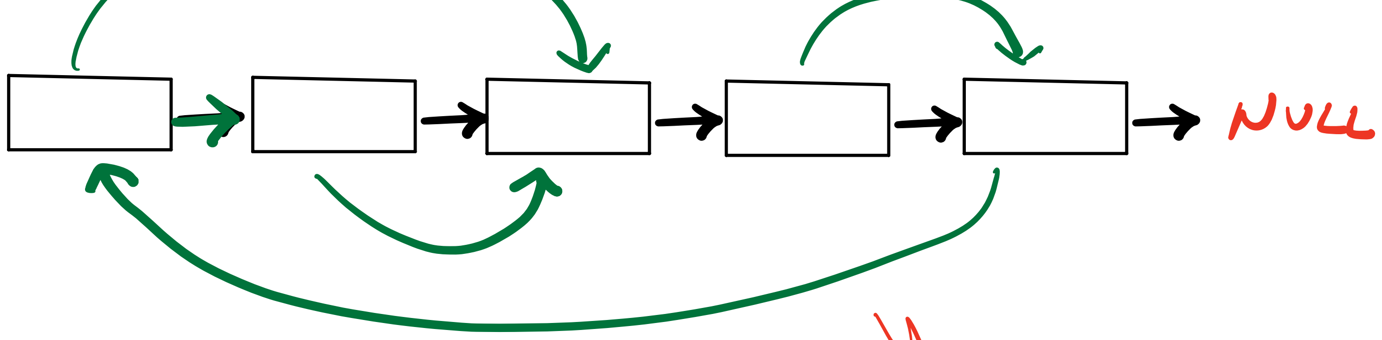
30 mins / Problem

Plan a session

$\Downarrow$

Interviews

Copy of a LL



Create a Deep Copy

```
class Node {
    int data
    Node next;
    Node random;
}
```

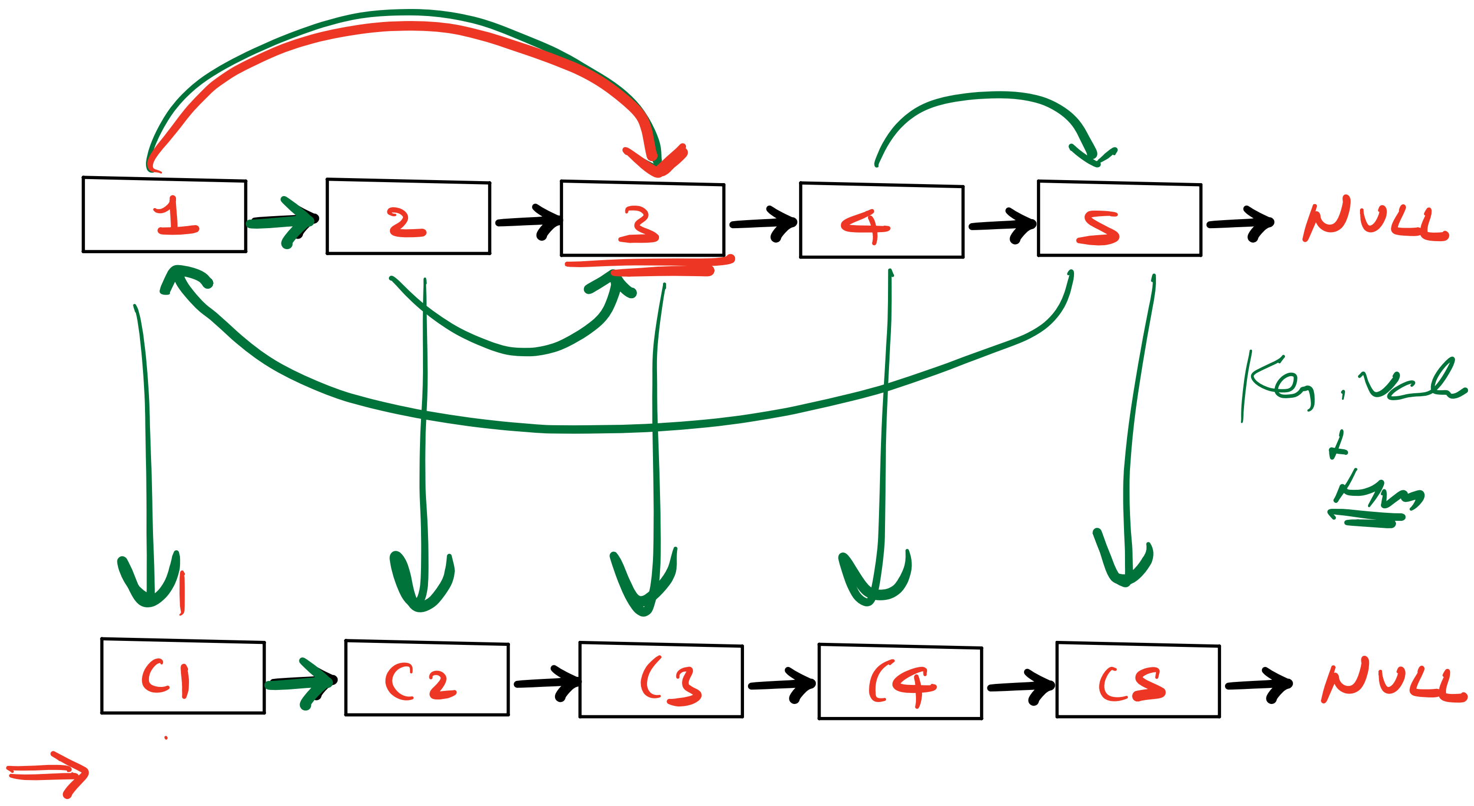
Shallow ~~vs~~ vs Deep

```
Node x
Node y = x;
```

{ Shallow copy }

D:// documents / case.xls ⟷ caseTemp.xlsx

Desktop:// caseShortcut.xlsx


1
2
3
4
5
NULL
C1
C2
C3
C4
C5
NULL
Key, value
&
Hm