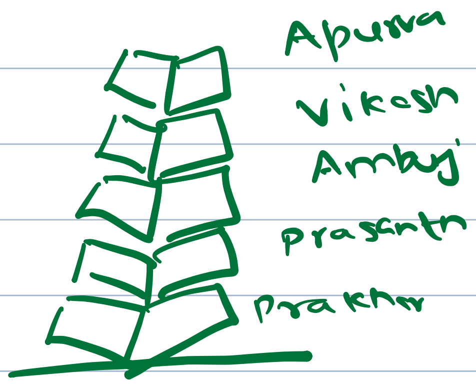
Agenda

1. What are Stacks?
2. Implementation of stack
    - Array
    - Linked List
3. Questions
    - Balanced Paranthesis
    - Double Character Trouble
    - Postfix Evaluation

Stack

① Linear data structure, store info in a sequence from bottom to top

② It follows LIFO
↓
Last In First Out



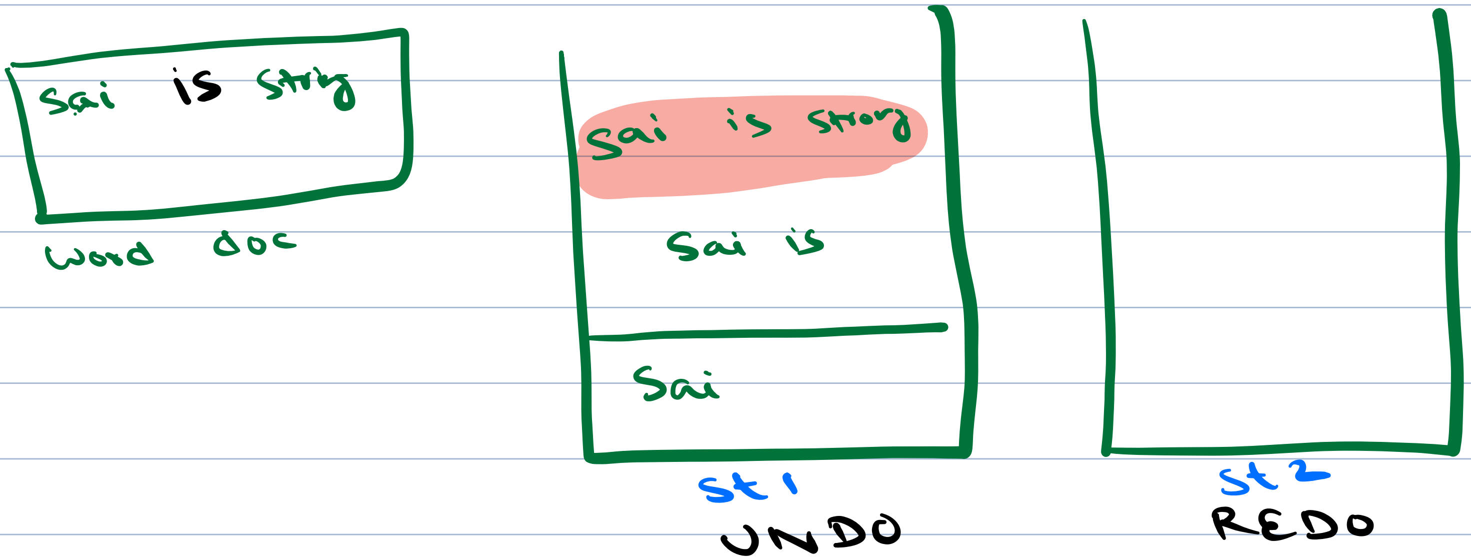
Elements can be accessed only from the top.

New elements added only at the top

Algorithms:

① Recursion

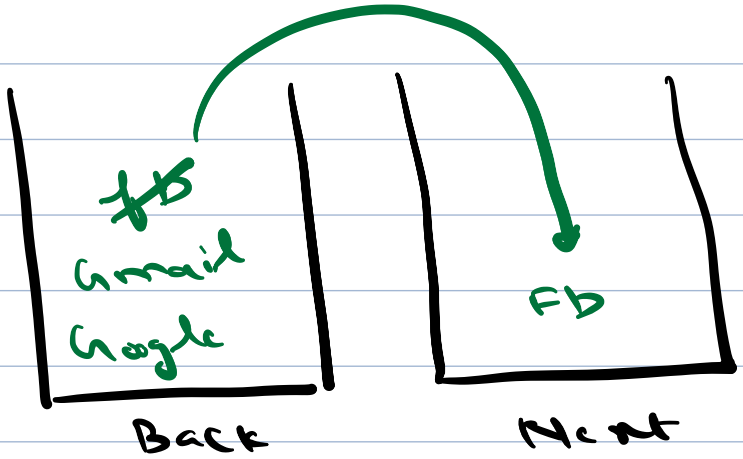
② Undo / Redo functionality



③ Browser next & back navigation

← →

Google → Gmail → Fb



④ Evaluate arithmetic Expressions

Operations on stack

1. Push → Push operation is to insert new element at top of the stack `void push(x)`

2. Pop → Remove an element from top of the stack `void pop()`

3. Top / Peek → Return the top element of the stack `data = top()`

4. is Empty → Check if stack is empty or not `boolean isEmpty()`

All operations are O(1) TC

Push(5) ✓

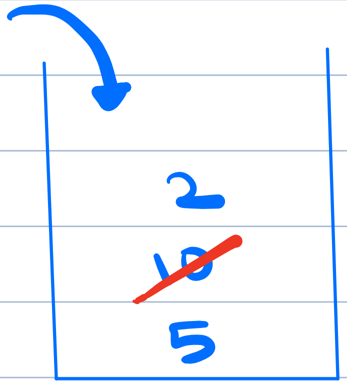
Push(10) ✓

Top() ✓ 10

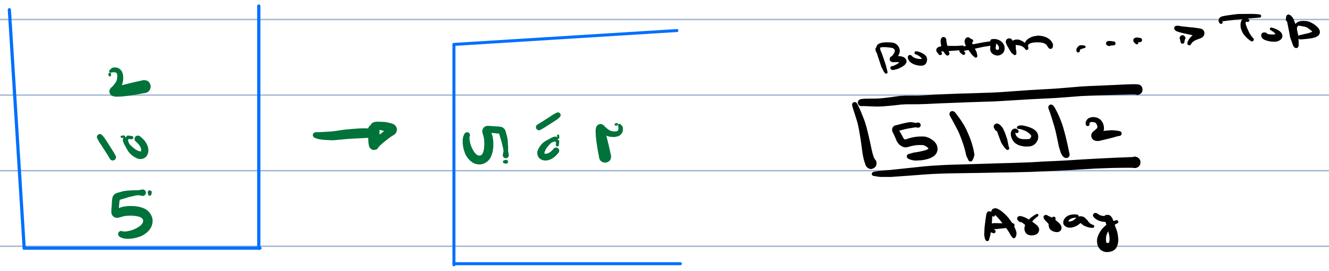
Pop() ✓

Top() ✓ 5

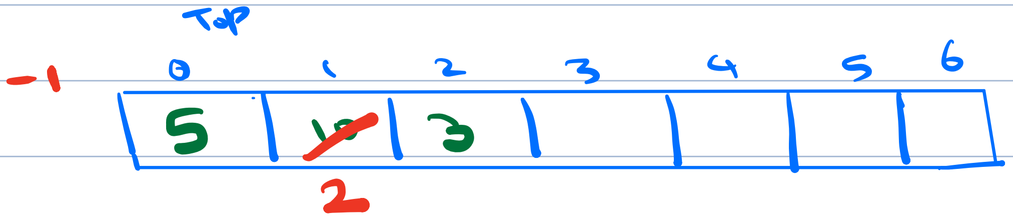
Push(2) ✓



Stack using arrays



Stack



Push(5) ✓

Push(10) ✓

Top() ✓ 10

Pop() ✓

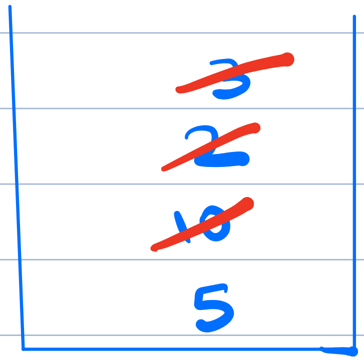
Push(2) ✓

Push(3) ✓

Top() ✓ 3

Pop() ✓

Pop() ✓



Stack → Array

Bottom to Top

0 to Top

Stack size = top + 1

```
class Stack {
    int [ ]  arr ;
    int  size ;   int  top

    Stack ( capacity ) {
        arr = new  Array (capacity)
        top = -1
        size = capacity
    }

    void  push (int  n) {
        if (top == size - 1)  print ("OVERFLOW")
        top ++
        arr [top] = n
    }

    void  pop() {
        if (top == -1)   print ("UNDERFLOW")
        top--
    }

    int  top() / peek() {
        if (top == -1)   print ("UNDERFLO W)
        return   arr [top]
    }

    bool   isEmpty() {
        return   top == -1
    }
```

1-7

```
Stack st = new Stack(10)
st.push(2)
print(st.top())
st.pop()
```

- Underflow
  Try to pop an element from an empty stack

- Overflow
  Try to push more elements when there is no space

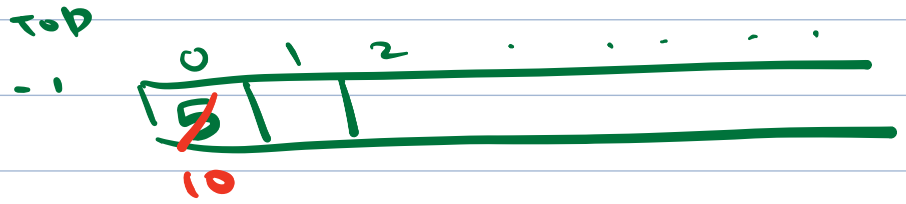
Problem with Implementation using array

① Array → Fixed size to create it

1000 opr → int ar[1000]

Push(5)

Pop()



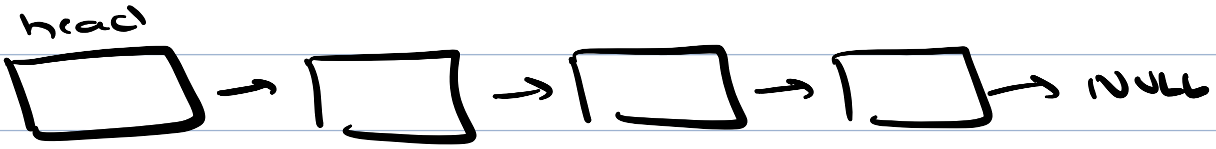
Push(10)

pop()

② Memory wastage

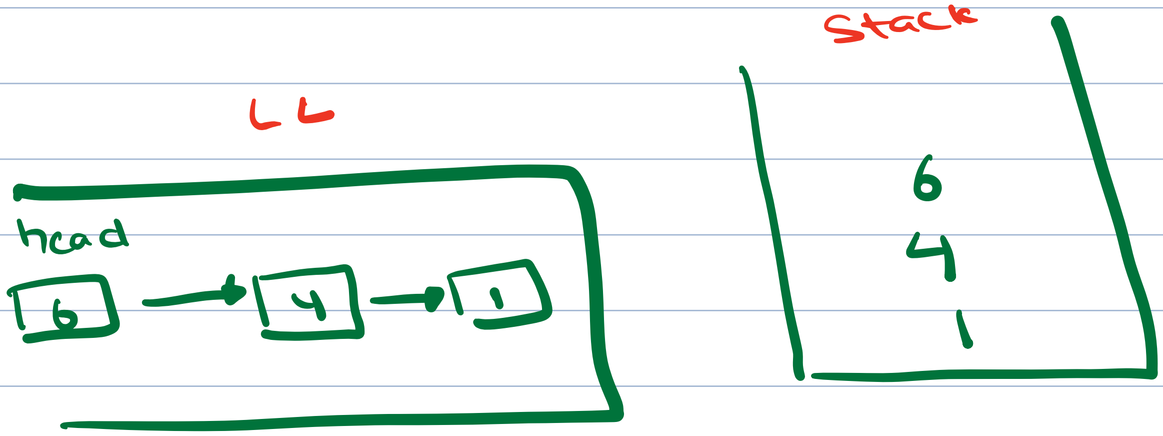
Implement stack using LL

Stack → Insertion and deletion at top (one end)



	Insertion	Deletion	
Head	O(1)	O(1)	✓

Tail	O(N)
	↓
	O(1)
	maintain tail
		O(N)	



head → top

Push(5) ✓

Push(10) ✓

Top() ✓ 10 .

Pop() ✓

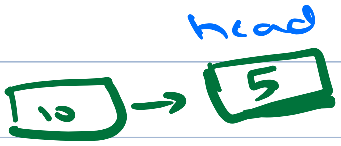
Push(2) ✓

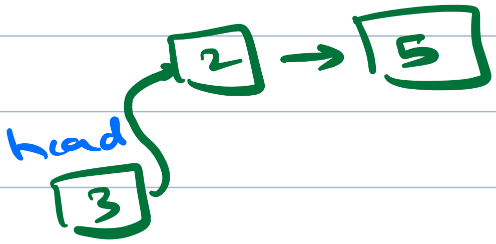
Push(3) ✓

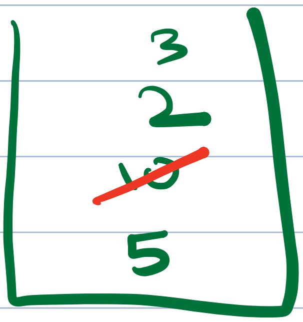
Top() ✓ 3

Pop()

Pop()







```
class stack () {

    Node head

    stack () {
        head = NULL
    }

    void push (int x) {
        Node nn = new Node (x)
         nn. next = head
          head = nn
    }

    void pop () {
        if (head = = NULL) {
        print ("UNDERFLOW")
        return }

          head = head. next
    }
```

C++

```
Node del = head
head = head.next
free (del)
```

```
int  top () {
    if (head = =NULL) {
    print ("UNDERFLOW")
    return }
    return   head.data
}

bool  isEmpty() {
    return   head ==NULL
}

}
```

10:32

Prob 3 : Check whether given sequence of paranthesis is valid ?   ( )   [ ]   { }

(( )) ✓

( ))) ( ) ✗

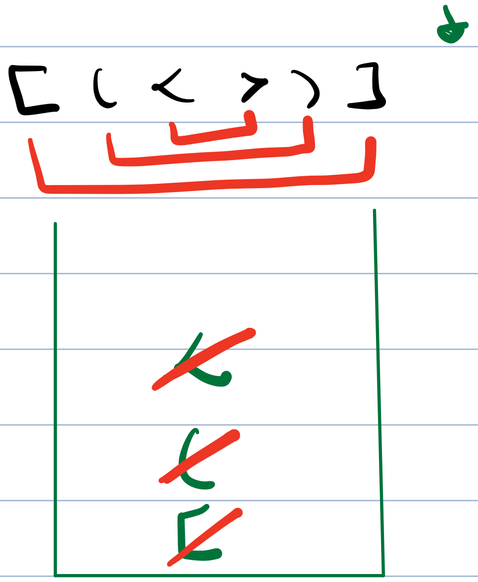
{ ( ) } ✓

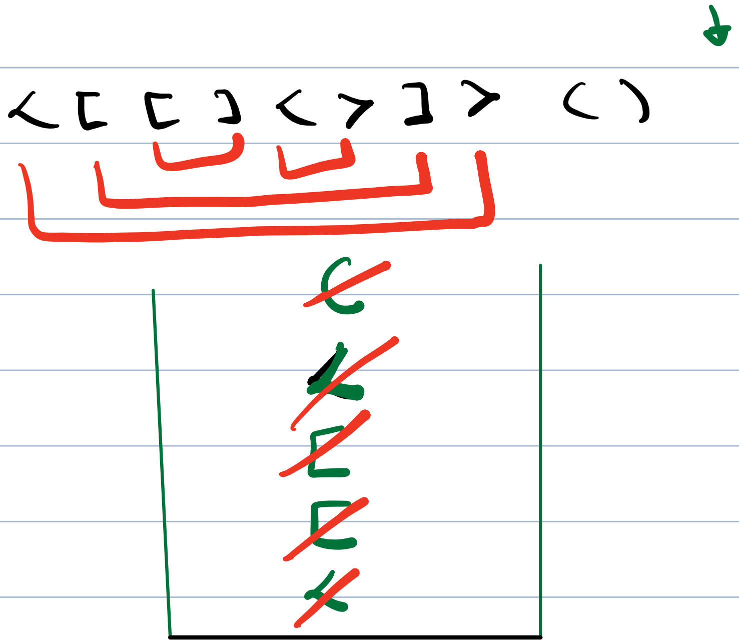
$\underbrace{[\underbrace{(\;)}]}\ \underbrace{\{\;\}}\ \underbrace{(\;)}$ ✓

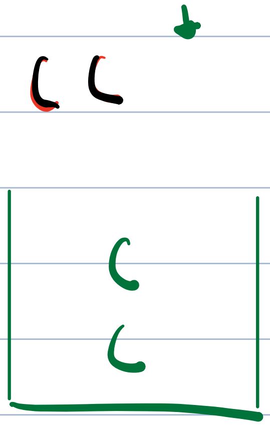
{ (}) ✗

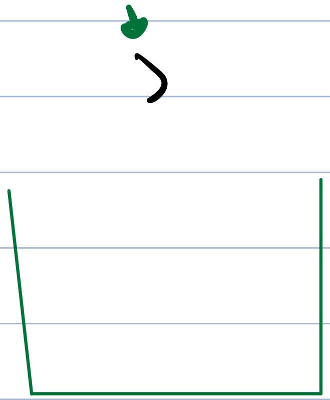
Approach : Push all opening brackets in stack. When we encounter closing bracket, check in stack whether it has corresponding opening bracket. If it is present, pop it.

Stack empty → balanced sequence









```
bool isValid (string seq) {
    stack <char> st
    for (i = 0 ; i < seq.length ; i++) {
        if (seq[i] == '(' || seq[i] == '[' ||
            seq[i] == '{') {
            st.push (seq[i])
        }
        else {
            if (st.isEmpty())
                    return false

            char open = st.top()
            if (seq[i] and open are same)
                    st.pop()
            else
                    return false
        }
    }
    return st.isEmpty()
}
```

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

```
① if ((cur == ']' && open == '[') ||
     (                            ) ||
      (                           ))
```

seq[i]

② switch (cur)

```
case ']' : return open == '['
case ')' : return open == '('
case '>' : return open == '<'
```

③ map<char, char> mp

>	:	<
]	:	[
)	:	(

```
if (open == mp[cur])
```

Prob 4: Given a string, remove equal pair of consecutive elements till it is possible.

I/P: abcddc

⇓

abcc

⇓

O/P : ab

abbcbbcacx

⇓

accacx

⇓

aacx

⇓

O/P : cx

cccc

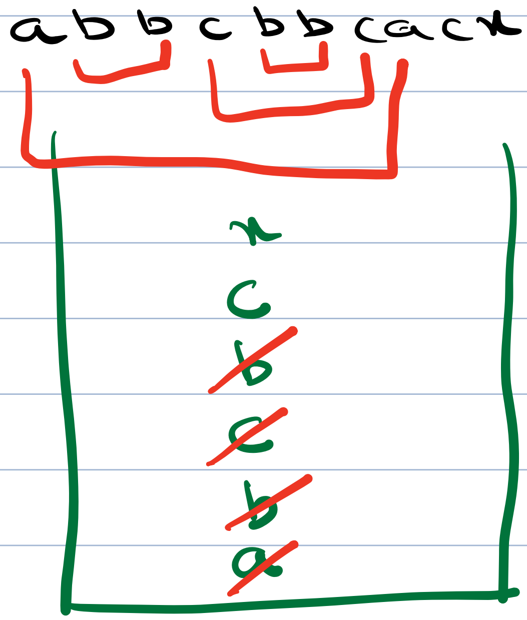
⇓

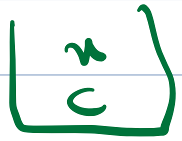
cc

⇓

O/P : " "

cbbc <( )>





ans = xc

↓ reverse

ans = cx

```
String removeEquals (String s) {
    Stack <char> st
    for (each char 'c' in s) {
        if (!st.isEmpty() && c == st.top())
                              st.pop()
        else {
                              st.push(c)
        }
    }
    String ans = ""
    while (!st.isEmpty()) {
        ans = ans + st.top()          st.pop()
    }
    reverse (ans)
    return ans
}
```

TC : O(N)

SC : O(N)

Prob 5 : Given a postfix expression, evaluate it

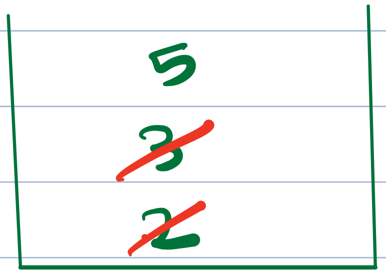
Infix Expression	Postfix Expression
2 + 3 op1 operator op2	2 3 +

[2 , 3 , +]

a operation b

operand → push

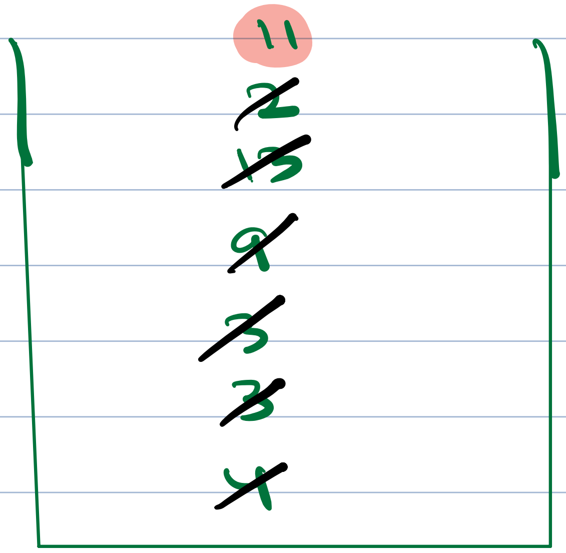
operator → action



b=3
a=2
a+b=5

4 3 3 * + 2 −

ans=11



b=2
a=13

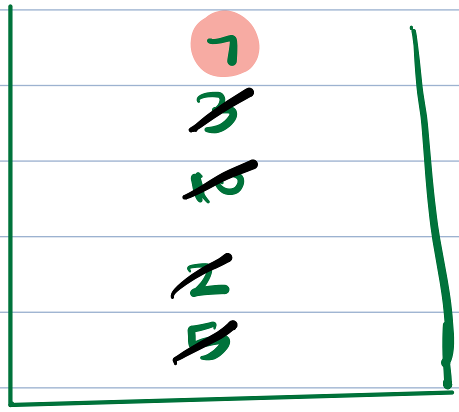
a−b=13−2
=11

b=3
a=3
a*b=9

b=9
a=4

[5, 2, *, 3, -]

ans=7



b=3
a=10

a−b=7

b=2
a=5
a*b=10

```
int evaluatePostfix (List<string> expression){
        stack <int> st
        for (ele in expression) {
            if ele is not an operator
                        st.push((int) ele)
            else {
                int b= st.top()        st.pop()
                int a = st.top()        st.pop()
                st.push( evaluate (a, b, ele))
            }
        }
        return st.top()
}
int evaluate (int opr1, int opr2, string operation)
{
        switch ( operation) {
            case '+' : return opr1 + opr2
            case '-' : return opr1 - opr2
            case '*' : return opr1 * opr2
            case '/' : return opr1 / opr2
        }
}
```

Tc : O(n)

Sc : O(n)