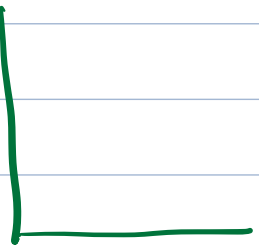
Agenda

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

Stack

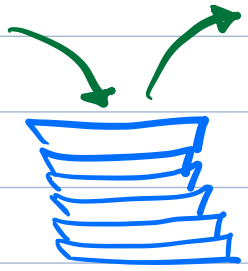
① Linear data structure, store info in a sequence from bottom to top



② It follows LIFO

↓

Last In First Out



Pile of plates / books

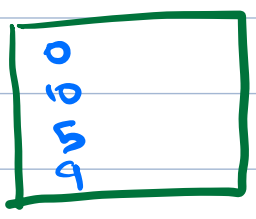
Elements can be accessed only from the top.

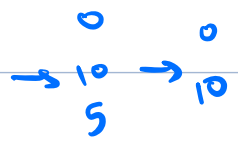
New elements added only at the top

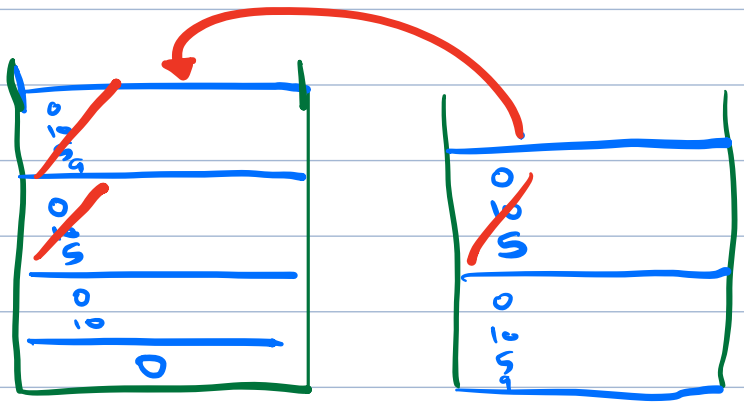
Algorithms:

① Recursion

② Undo / Redo functionality





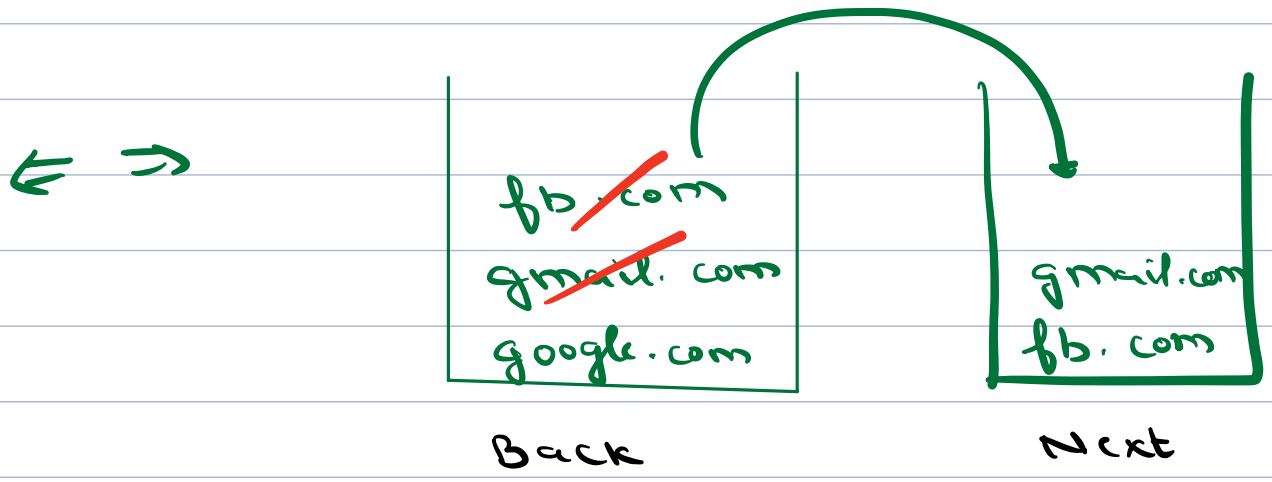


Stack → doc state

St 1 UNDO

St 2 REDO

③ Browser next & back navigation



④ Evaluate arithmetic Expressions

Operations on Stack

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

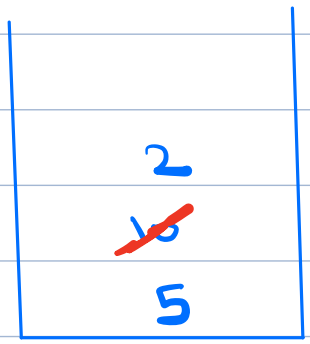
2. Pop → Remove an element from top of the stack `void pop()`

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

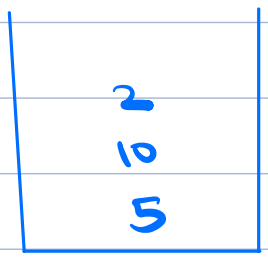
4. isEmpty → Check if stack is empty or not `boolean isEmpty()`

All operations are O(1) TC

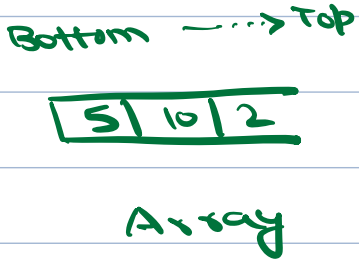
Push(5) ✓
Push(10) ✓
Top() ✓ 10
Pop() ✓
Top() ✓ 5
Push(2) ✓



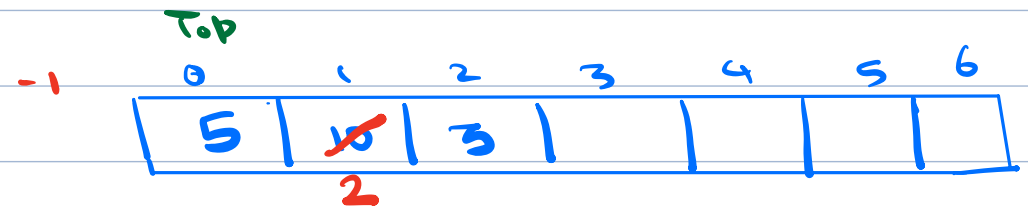
Stack using arrays

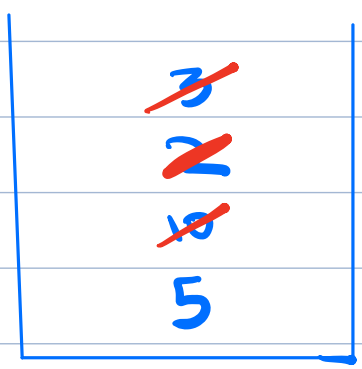


Stack



Push(5) ✓
Push(10) ✓
Top() → 10
Pop() ✓
Push(2) ✓
Push(3) ✓
Top() → 3
Pop() ✓
Pop() ✓





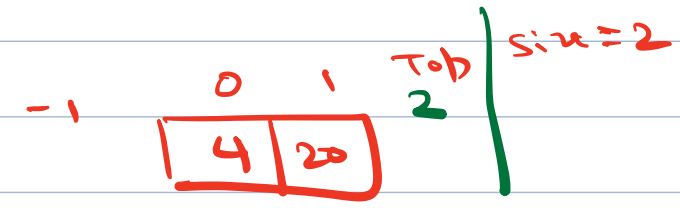
Array → Bottom to Top
0 to Top

```
class stack {

    int[] arr;      int size;
    int top;

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

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

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

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

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

```
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



push(4)

push(20)

push(3) ✗

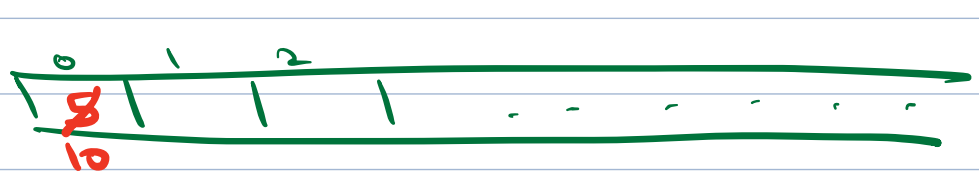
Problem with Implementation using array

① Array → Fixed size to create it

1000 ops → int arr[1000]

push(5) ✓

pop() ✓



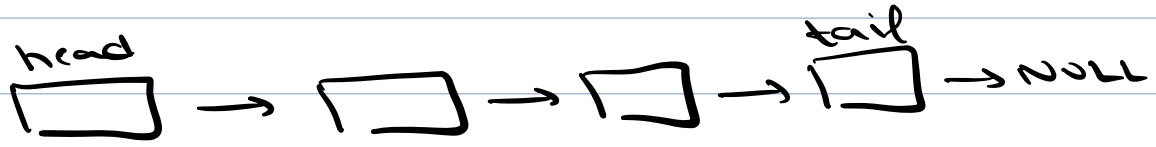
push(10) ✓

pop() ✓

push

② Memory wastage

Implement stack using LL

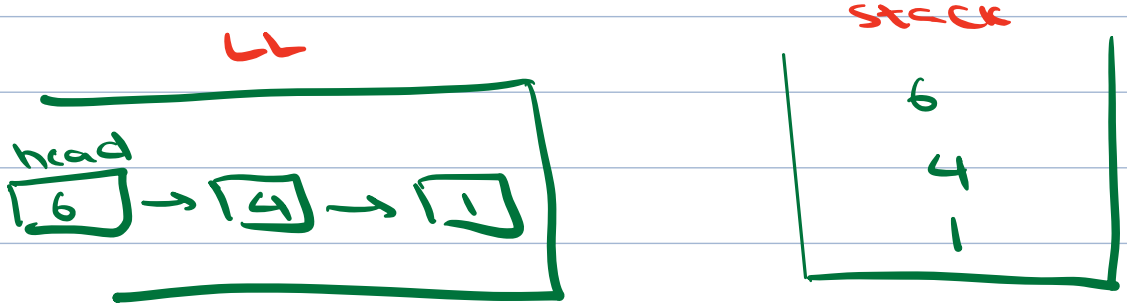


Insertion and Deletion at head or tail ?

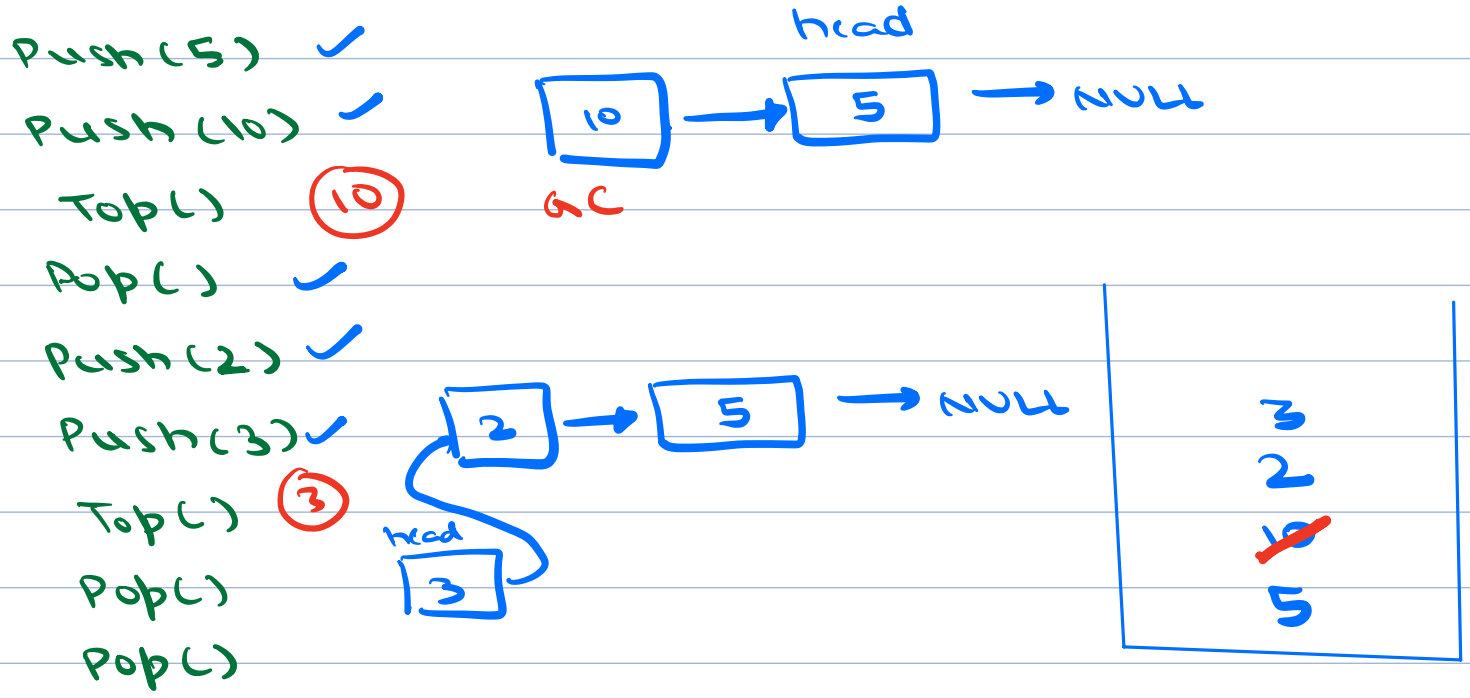
Head

Insertion at tail ? O(1)

Deletion at tail ? O(n)



head → top




```
class Stack () <

    Node head;

    Stack () <
    |   head = NULL
    >

    void push (int x) <
    |   Node newNode = new Node (x)
    |   newNode.next = head
    |   head = newNode
    >

    void pop() <
    |   if (head == NULL) print ("UNDERFLOW)
    |   head = head.next
    >
```

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

bool isEmpty() {
    return head == NULL
}
}
```

10:27

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

(( ))

( ))) ( )

< ( ) >

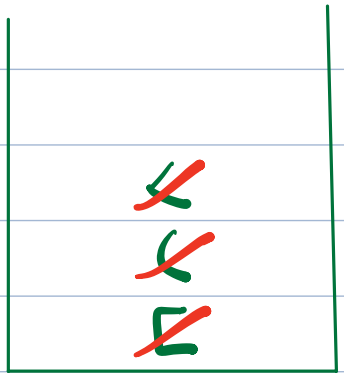
< (>)

[( )] < > ( )

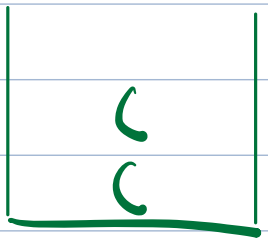
Approach : Push all opening brackets in stack. When we encounter closing bracket, check in stack whether it has corresponding opening bracket. If it is present, pop it.

Stack empty → balanced sequence

[(< >)]

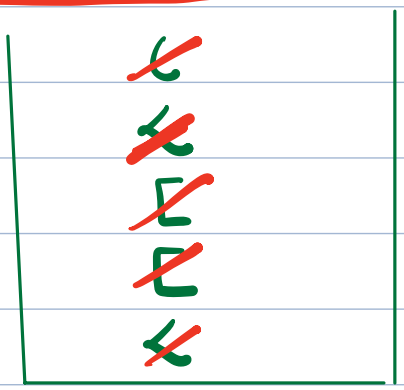


((



Stack not empty → false

< [[ ] < >] > ( )



)

False

```
bool isValid (String sequence) {
    Stack <char> st
    For each char in sequence {
        if char is opening parenthesis
            st.push(char)
        else {
            If the st is empty()
                return false

            char open = st.top()
            If opening and cur are same
                st.pop()
            else
                return false
        }
    }
    return st.isEmpty()
}
```

TC : O(n) SC : O(n)

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

② switch (cur)

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

③ map <char, char> mp

>	<
]	[
)	(

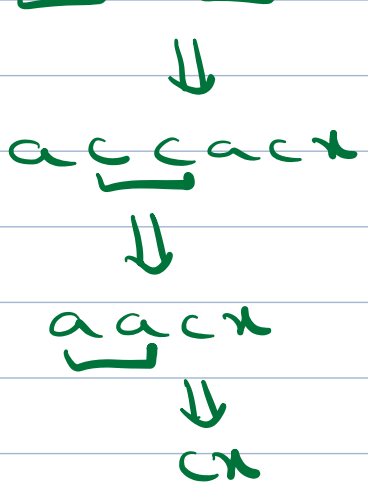
if (open == mp[cur])

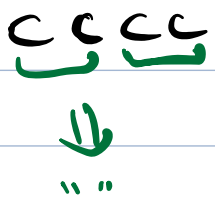
Prob 4: Given a string, remove equal pair of consecutive elements till it is possible.

I/P: abcddc

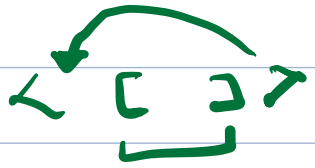
O/P: ab

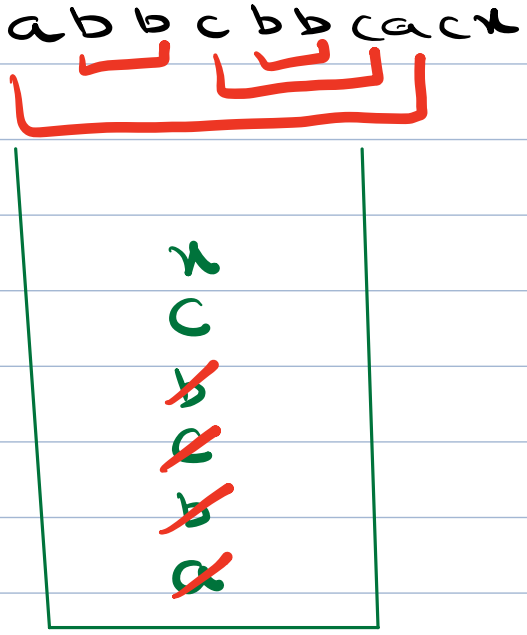
abbcbbcacx





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 += st.top()    st.pop()
    }
    reverse(ans)
    return ans
}
```

TC: O(n)
SC: O(n)

Prob 5: Given a postfix expression, evaluate it

Infix Expression

2 + 3 → 2 3 +

Opr1 operator Opr2 → Op1 Op2 operator

Postfix Expression

4 + 3 * 3 - 2

4 + 9 - 2

13 - 2

11

4 + (3 * 3) - 2

4 + 3 3 * - 2

4 3 3 * + - 2

4 3 3 * + 2 -

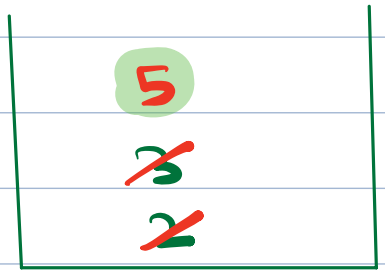
[2, 3, +]

ans = 5

operand → push

operator → action

ans → st.top()



x = 3

y = 2

3 + 2 = 5

4 3 3 * + 2 −

x = 3 y = 3 x*y = 9

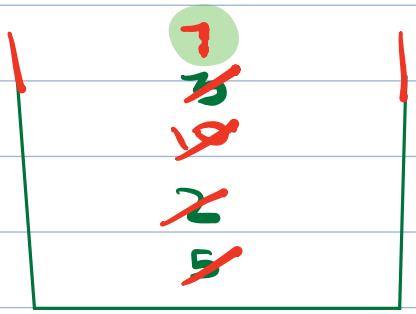
x = 9 y = 4 x+y = 13

x = 2 y = 13 y − x = 13 − 2 = 11

ans = 11

---

5 2 * 3 −



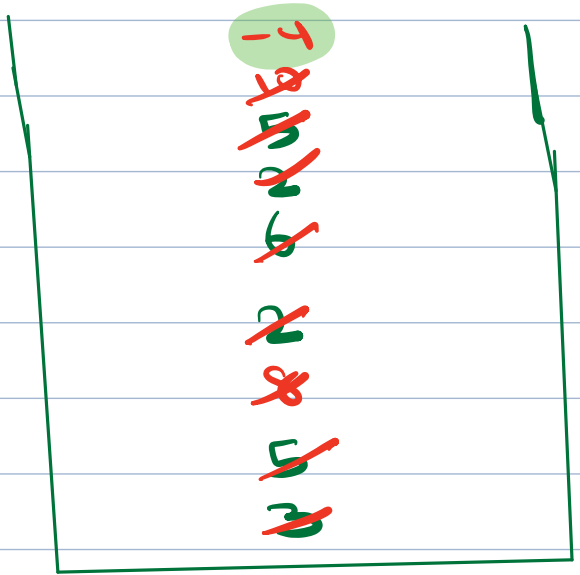
ans = 7

x = 2 y = 5 y * x = 10

x = 3 y = 10 y − x = 7

---

3 5 + 2 − 2 5 * −



ans = -4

x = 5 y = 3 y + x = 8

x = 2 y = 8 y − x = 6

x = 5 y = 2 y * x = 10

x = 10 y = 6 y − x = −4

```
int evaluatePostfix (List<String> expression){
    Stack <int> st
    for ele in expression {
        if ele not an operator
            st.push ( int (ele))
        else {
            int x = st.pop()
            int y = st.pop()
            st.push(evaluate (y, x, 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)

Doubts

H = 32

20 10 4 3 1

H = 32

cnt = 0 → 1 → 2 → 3

H = 32 → 12 → 2 → -2

9 9 4 4

H = 32

↓ -9

23

↓ -4

19

↓ -9

10

⋮

Tree Map

4	x1
9	x1

lastWeapon = 9

↓

4