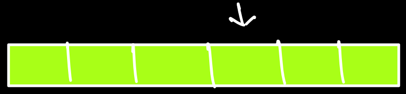
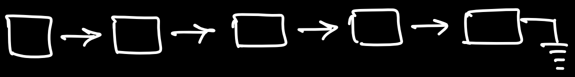
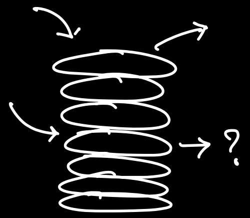
Stack.

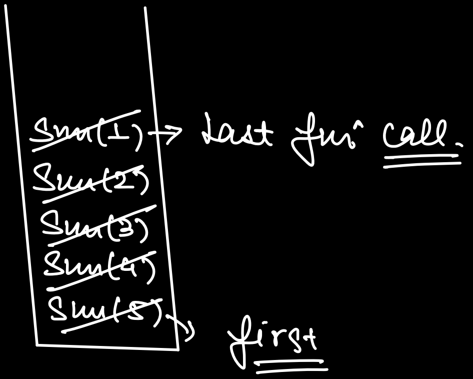
→ Arrays ⇒



→ Linked List ⇒







→ Recursion Call Stack

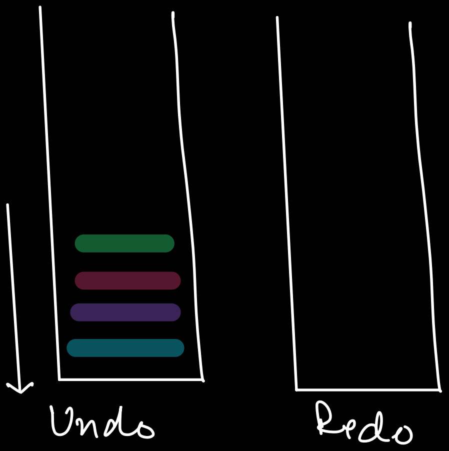
Last In, First Out.

⇒ Stack.

① Recursion Call Stack

② Web browsers.

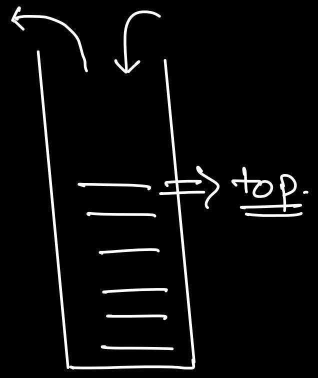
③ Undo | Redo



→ **Abstract** Data Type (ADT)

Stack as an ADT
↓
LIFO

TC: O(1) {
- push(x) ☆
- pop() ☆
- top() / peek()
- size()
- isEmpty()
- Clear()



Stack Implementation :-

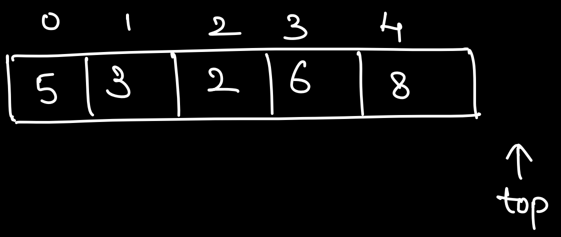
① Array

② LL.

* Array implementation of Stack.

```
int arr[5];
top = -1;
```

↓
Stack is Empty.



0	1	2	3	4
5	3	2	6	8

- push(5) :

→ top++
→ arr[top] = 5 } ⇒ O(1)

- push(3), push(2), push(6), push(8),

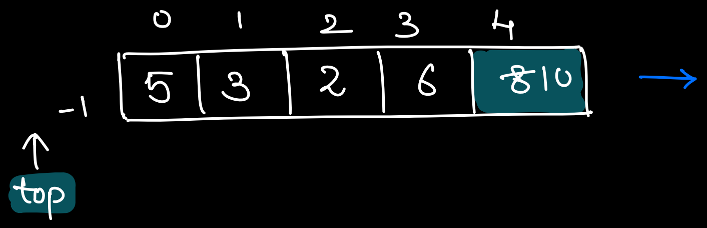
push(1)

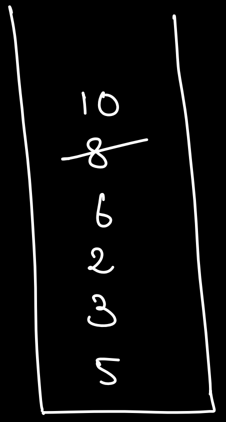
↳ Array Index Outof bound

→ Overflow. ✓

⇒ Dynamic Arrays.

Pop() : if (top > -1) top--





push(10) →

pop()

pop()

pop()

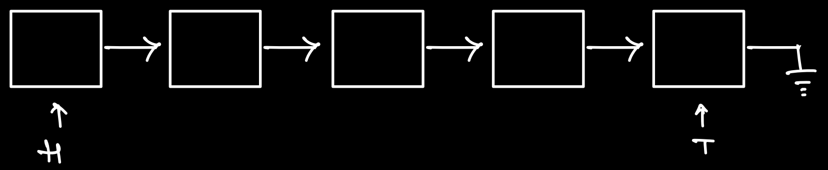
pop()

pop()

pop()

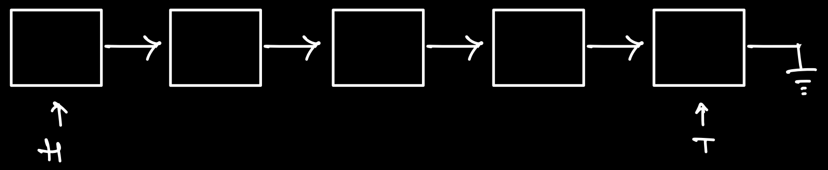
⇒ Underflow.

Linked List implementation of Stack.



TC of insertion at tail ⇒ O(1)

TC of deletion at tail ⇒ O(N)



TC of insertion at head ⇒ O(1)

TC of deletion at head ⇒ O(1)

push(x)

pop() } $\boxed{O(1)}$

```
Stack<Int> st = new Stack<>(); → Java
Stack<int> st; → C++
St.push(x);
St.pop();
```

* Q. Given the library stack, Create a new Stack like DS that gives us :

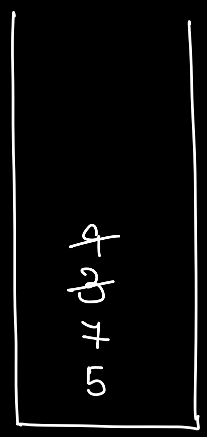
push(x)
pop()
getMin()

TC : O(1)

newStack :

```
push(5)
push(6)
push(4)
push(7)
getMin() → 4
pop()
pop()
getMin() → 5
push(3)
getMin() → 3
pop()
getMin() → 5
```

* push(5)
push(7)
push(3)
push(9)
getMin() → 3
pop()
getMin() → 3
pop()
getMin() → ③ × X
└→ 5

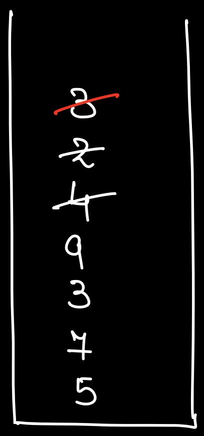


min = ~~ϕ~~ ~~5~~ 3

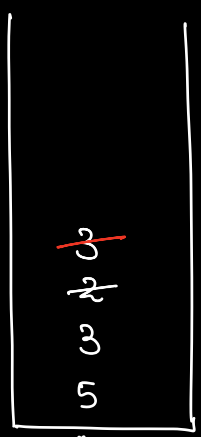
minStack.

#

push(5)
push(7)
push(3)
push(9)
getMin() ⇒ ③
push(4)
push(2)
getMin() ⇒ ②
pop()
getMin() ⇒ ③
pop()
getMin() ⇒ ③ ✓
push(3)
pop()



Stack



minStack

```
push(x) :   St.push(x)
O(1)        if (x <= minStack.top()){      → Check if stack isn't Empty.
                 minStack.push(x);
            }

pop() :   { x = st.top();
O(1)        St.pop()
            if(x == minStack.top()){
                 minStack.pop();
            }

getMin() ⇒ minStack.top()
O(1)
```

Q. Given a string, Remove pairs of consecutive elements repeatedly until there are NO duplicate pairs.

Amazon
MS.

S: a b c d d c b k

a b c c b k

a b b k

a k.

Quiz

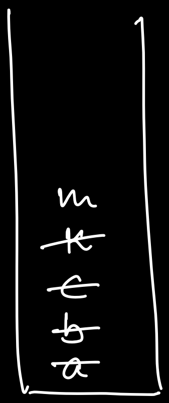
S: a a a b

ab.

Quiz

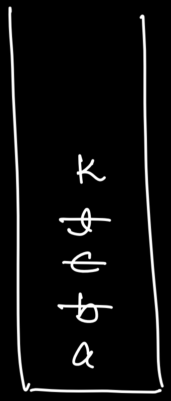
S: abckkcbam

↑



S: abcddcbk

↑



ka

reverse.

* Expression Evaluation.

× /
+ −

$\underbrace{7 \times 1} + 2 - \underbrace{8 \times 3} + \underbrace{10/5} \Rightarrow$

$7 + 2 - 24 + 2 = -13$

Computers uses

Infix Notation	Postfix Notation
A + B	A B +
A − B	A B −
A × B	A B ×
A / B	A B /

* $10 + 3 \times 4 \Rightarrow 10 + \underbrace{3 \times 4}$

$= 10 + 3\,4\,\times$

$= 10\ 3\,4 \times +$

- Infix to Postfix ✓
- Evaluate the postfix

# $\underline{\underline{A + B \times C}}$

$\Downarrow$

$A + BC\times$

$\Downarrow$

$A\ BC \times +$

$\underline{\underline{A \times B + C}}$

$\Downarrow$

$AB\times + C$

$\Downarrow$

$AB \times C +$

- Infix to Postfix

Quiz

4 + 8 * 7

↓

4 + 8 7 x

4 8 7 x +

Quiz

10 + 3 x 4 - 7

↓

10 + 3 4 x - 7

10 3 4 x + - 7

10 3 4 x + 7 -

Quiz

10 / (4 - 2) * 6 + 9

↓

10 / 4 2 - * 6 + 9

↓

10 4 2 - / * 6 + 9

↓

10 4 2 - / 6 * + 9

↓

10 4 2 - / 6 * 9 +

Quiz

$$(10+3) * 2 - (7-6) * (4+8)$$

$$10\ 3+ \quad * \quad 2 \quad - \quad 7\ 6\ - \quad * \quad 4\ 8+$$

$$10\ 3+\ 2\ * \quad - \quad 7\ 6-\ 4\ 8+\ *$$

$$10\ 3+\ 2\ *\ 7\ 6-\ 4\ 8+\ *\ -$$

⇒ Operands follows the same relative order b/w infix & postfix notation.

$$10 + 3 \longrightarrow 10\ 3\ +$$

↑

$$\boxed{+ \qquad\qquad}$$

$$4 + 8 * 7 \longrightarrow \underbrace{4\ 8\ 7\ \times\ +}$$

↑

Box $\boxed{+\ \times \qquad\qquad}$

$+ \longrightarrow \times$ } precendence

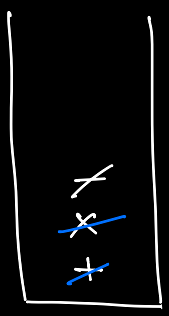
$$4 \times 8 + 7 \longrightarrow 4\ 8\ \times\ 7\ +$$

↑

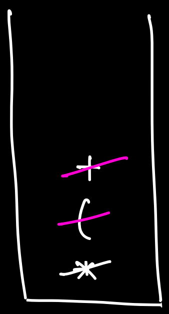
$$\boxed{\cancel{\times}\ + \qquad\qquad}$$

⇒ STACK.

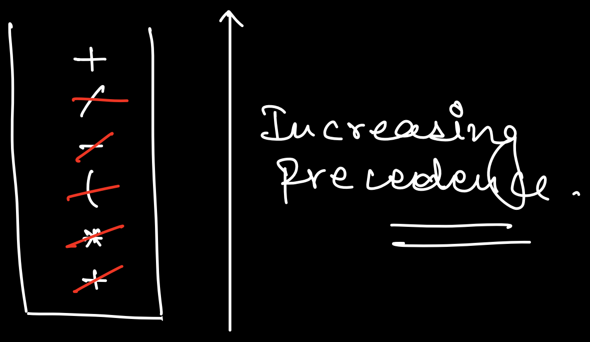
$10 + 3 \times 4 - 7 \longrightarrow 10\ 3\ 4\ \times\ +\ 7\ -$



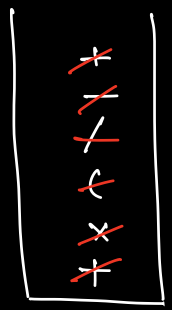
$10 * (3+4) \longrightarrow 10\ 3\ 4\ +\ *$

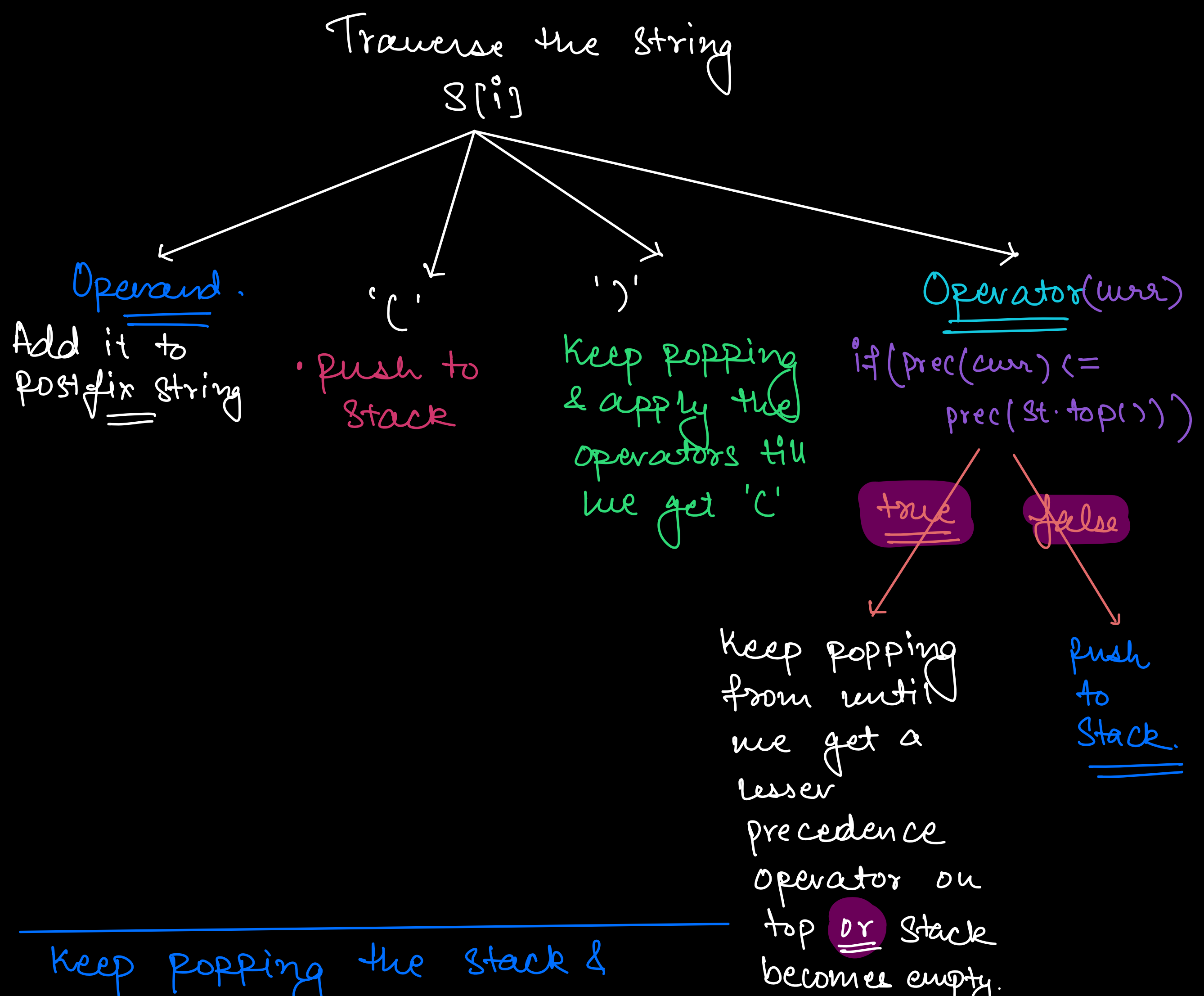


$3 + 10 \times (3 - 4/2) + 3 \longrightarrow 3\ 10\ 3\ 4\ 2\ /\ -\ *\ +\ 3\ +$



$3 + 10 \times (4/2 - 3) + 3 \longrightarrow 3\ 10\ 4\ 2\ /\ 3\ -\ \times\ +\ 3\ +$





Traverse the string

S[i]

Operand.
Add it to postfix string

'('
• Push to Stack

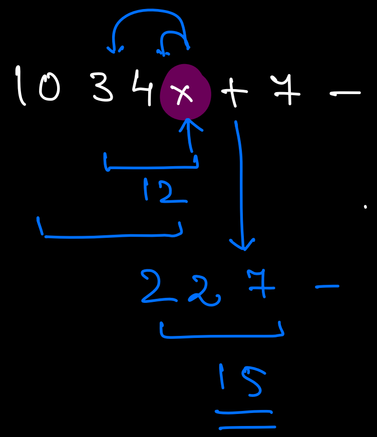
')'
Keep popping & apply the operators till we get '('

Operator(curr)
if(prec(curr) <= prec(St.top()))

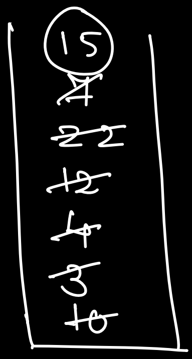
true
Keep popping from until we get a lesser precedence operator on top or Stack becomes empty.

false
Push to Stack.

Keep popping the stack & apply until Stack is Empty.



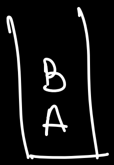
10 3 4 × + 7 −



A B op

A op B

A − B ⇒ A B −



A - B