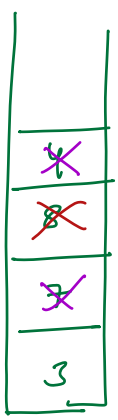
Today's Content :

- → Stacks Basics
- → Linked List Implementation
- → Balanced parantheses
- → getMin() / Without Extra Space
- → k^{th} repeating characters

// Stacks : LIFO → 1) Stack of plates
Last in first out
2) Idly plates

Ex: 3 7 8 top() pop() 4 pop() pop() top()
8 we delete 8 4 7 3



operations:
- push(x)
- top() : Return top ele
- pop() : Top elem gets rem
- size() : No: of elements in st
- isEmpty() :

Real world
- → undo/redo
- → Browsing ←, →
- → Browsing History
- → Functional Calls
- → Call logs
- → { Evaluating : Expression } →

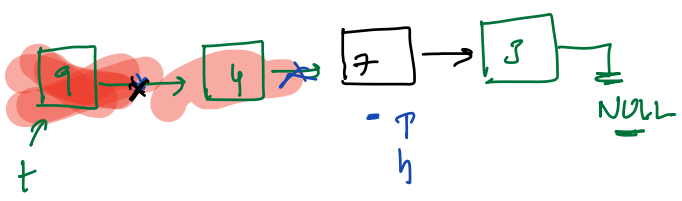
// Implementation of Stack with linked list

```
class Node {
    int data
    Node next;
    Node(n) {
        data = n
        next = NULL
    }
}
```

// Say no double linked list

// obs: { Delete at head in SLL : O(1)
Insert at head in SLL : O(1) }

adv



Ex: 3 7 8

Insert 3 at head Node

top() → 8 → h.data

pop() → we delete 8

4 9

pop() → 4
pop() → 7
top() → 3 → h.data

```
Node t = h
h = h.next
t.next = NULL
free(t)
```
↳ It will free that memory of t

TC:

push(x)	O(1)
top()	O(1)
pop()	O(1)
size()	O(1)

we can simple take a size variable & update it accordingly

Note: Say linked list is Empty : (h == NULL) && (size == 0)

top() } // not possible
pop() } // We will get error.

// In built libraries of Stack. → C++
→ Java
→ your language of choice
→ You can google search & get in your choice. {Vimp}

// Stack < int > st → I have declared Stack. st
↳ Every element should be int

- st.push(n) →
- st.pop() → delete top element
- st.top() // .peek() → Returns top Elem
- st.size() → Number of Elements

→ O(1)

Simple:

getMin() → we have create 1 more function to get minimum of Main Stack.

↳ O(1)

3 8 gm() 2 4 gm() pop() pop() gm() pop() 3 pop()

gm() → 3

gm() → 2

pop() → St1.pop() St2.pop()

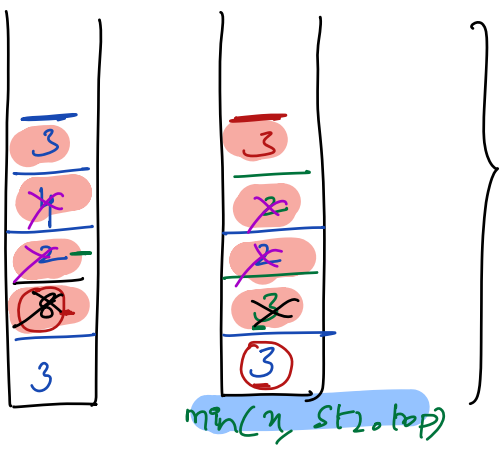
St.peek()

gm() → 3

Idea 2: Stack1 keep data, Stack2 min val till now

Stack<int> S1, S2; → we keep min

Edge Case work



O(1) top() { return S2.top() }

pop() { S1.pop(), S2.pop() }

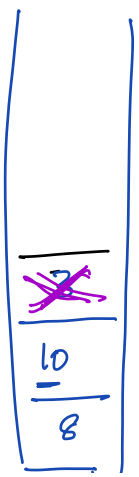
SL: O(N) push(x) { S1.push(x), S2.push(min(x, S2.top)) }

getMin() { S1.getMin() }

Ex: 8 10 3 getMin() pop() getMin() ?

↳ 3

Idea: Take a min variable & update it?



min_val = 3

min_val = 8

min_var = 8

When we pop(), update minval is not possible

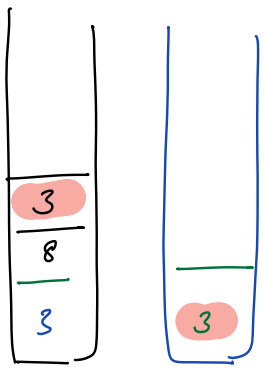
↳ you are loosing previous min

Idea 2:

→ Since min not changing let's not insert again?

// Eg: 3 (8) (3) pop()

remove 3 from both



// Without using any extra stack → minV

pop() pop() pop() pop()

↑ ↑ ↑ happy

removal	inserting		
x: 9	9	9	minV = 0 * pop
x: 0	0(2) - 2	: -2	minV = {0} → 2minV - (top) : 2(0) - (-2) = 2
x: 6	6	: 6	minV = 2 → minV = 2
x: 2	2(2) - 3	: 1	minV = 2 → minV = (2)(minV) - top) = (2)(2) - 1 = 3
x: 10	10	: 10	minV = 3 → minV = 3
x: 3	2(3) - 5	: 1	minV = 3 → 2*minV - top = (2)(3) - 2 = 5
x: 8	8	: 8	minV = 5 → minV = 5 =
x: 5	5	: 5	minV = 5

```
push(x){
  // x new element
  if(x >= minv){
    // minv is not changing
    st.push(x)
  }
  else // x < minv
    x - minv < 0
    2x - minv < x
    st.push(2x - minv)
    minv = x;
    // x is new min
  }
}
```

```
pop(){
  int x = st.top();
  if(x >= minv){
    st.pop()
    minv not change
  }
  else
    minv = 2*minv - x;
    st.pop()
  }
}
```

⇒ O(1)

```
getMin(){
  return minv
}
```

Edge: Stack Empty

```
top(){
  int x = st.top()
  if(x >= minv){
    return x
  }
  else return minv }
}
```

// x is new val, minV = pmin

if (x < minV) {

  Insert (2x - pmin)

  minV = x

}

// top of stack = (2x - pmin)

// minV = x

// pop

y = st.top

if (y < minV) {

  y = (2x - pmin)

  minV = x

  2*minV - y = pmin

  2(x) - (2x - pmin)

}

= (2x - 2x + pmin

2x

// Remove k consecutive character : TC: O(N) (linear) SC: O(N)

k = 3

Str: a c b b b c d d d c e a

a c c c e a

a e a → (Final String)

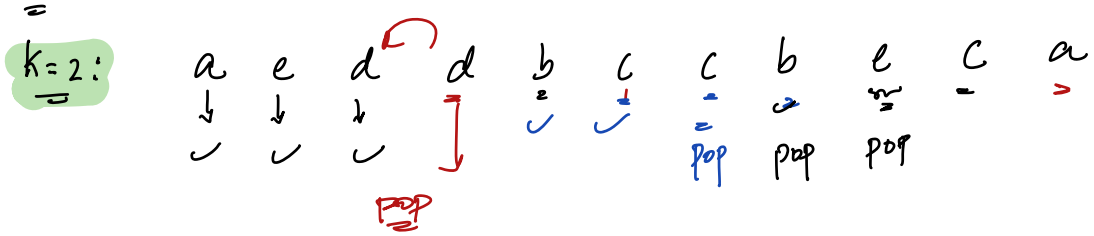
k = 2 :

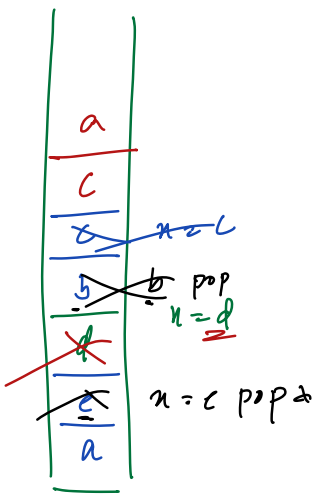
a e d d b c c b e c a

a c a

Eg: → {Balanced parantheses}

k=2:





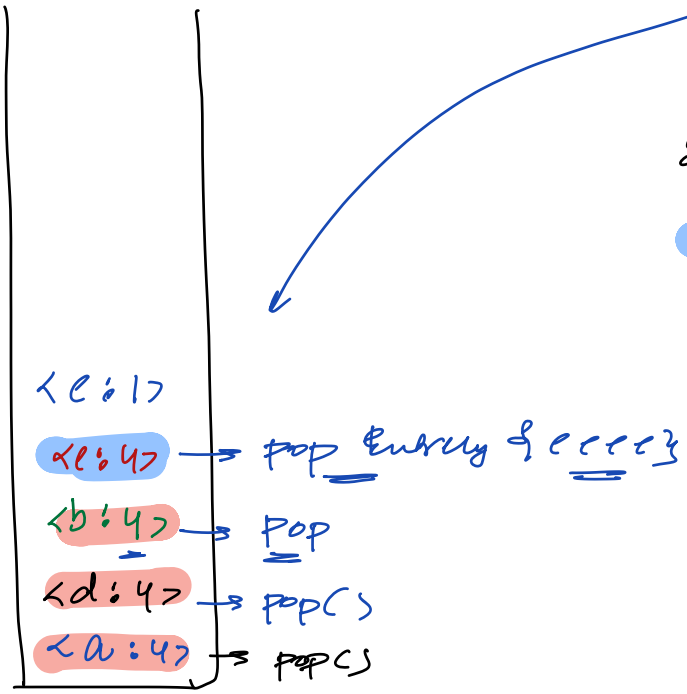
k=2: Idea:

→ If new char ch comes

```
if (ch == st.top()){
    st.pop();
}
else{
    st.push(n)
}
```

k = 4 : { a a d b b b e e e e b d d d a a e }

count of how many consecutive characters come till now.?



pair<ch, int>

Stack < pair < ch, int >> st

If new char ch comes

```
If ( ch == Top_character) {
    // increase freq by 1
    if freq == k
     pop()
}
else
    st. push ({ch, 1})
}
```

{ // What is left in Stack
  That is your ans }