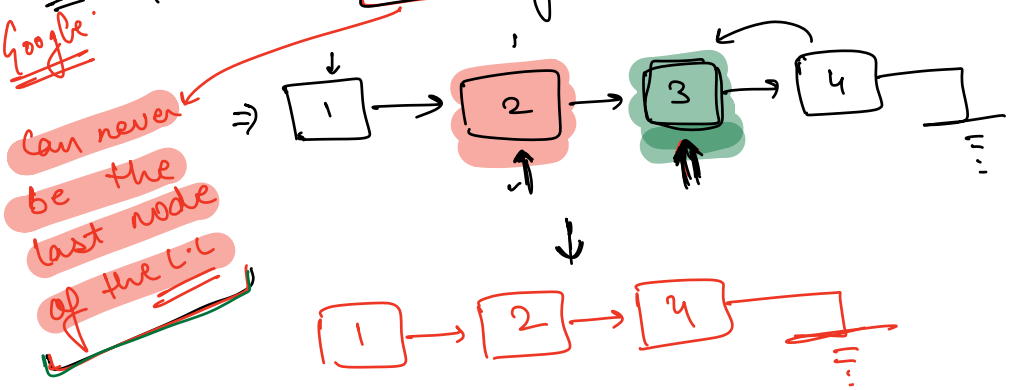
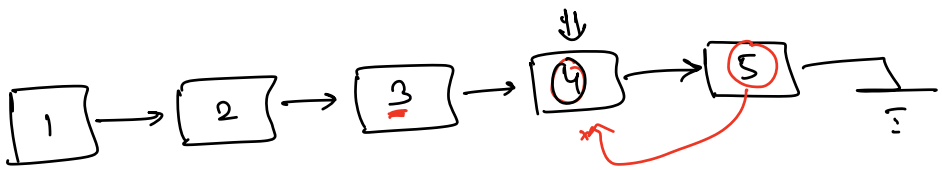
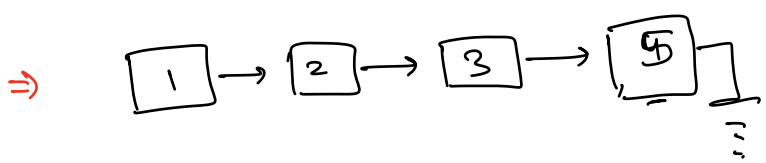
Q Given a node of a L·L, Delete this node from L·L

Google.

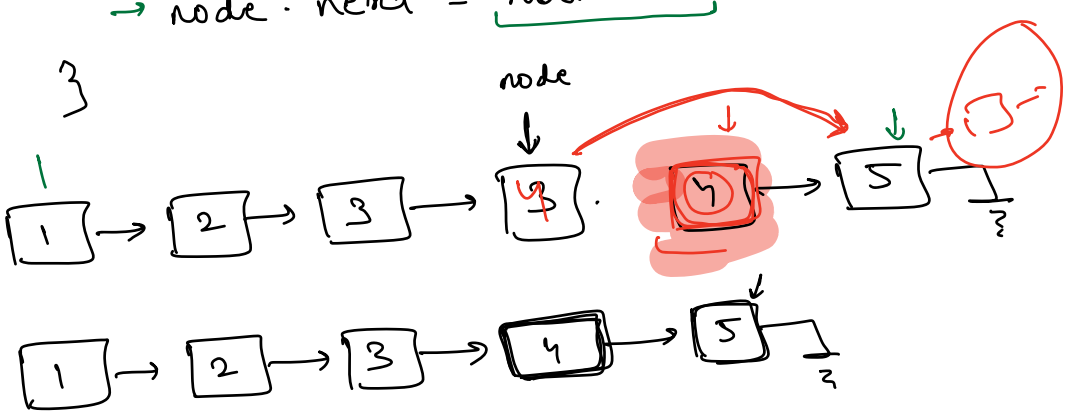




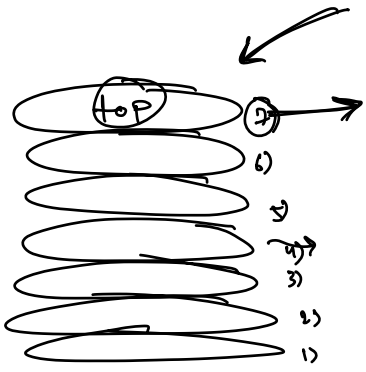



```
void deleteNode(Node node){
    node.data = node.next.data;
    node.next = node.next.next;
}
```

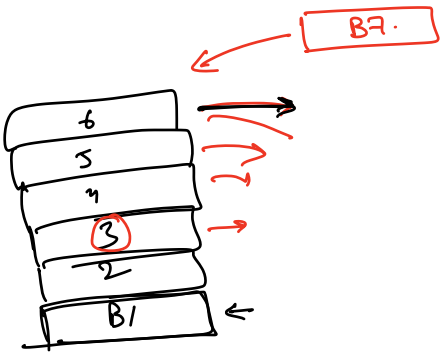
TC: O(1)
SC: O(1)



Edges cases :- 1) size of L·L = 1. X
2) Given node is 1st node.
3) If the given node is last node.
↳ NPE.



File of plates.



* Insertion or Deletion both from one end only.

⇒ Last In, first Out.

⇒ Abstract Data Type :- (ADT)

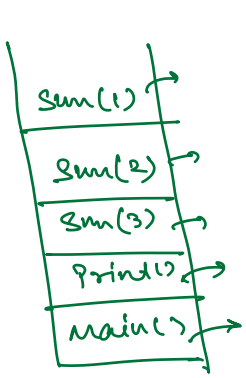
Stack :-

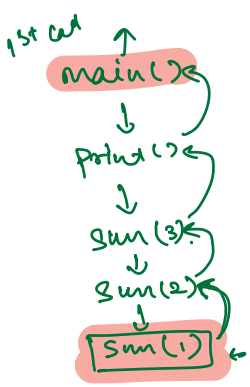
- push(x) (Insert) } Mandatory op.
- pop() (delete)
- top() (peek) } optional.
- isEmpty()
- size()

Applications :-

1) Recursion call Stack :-

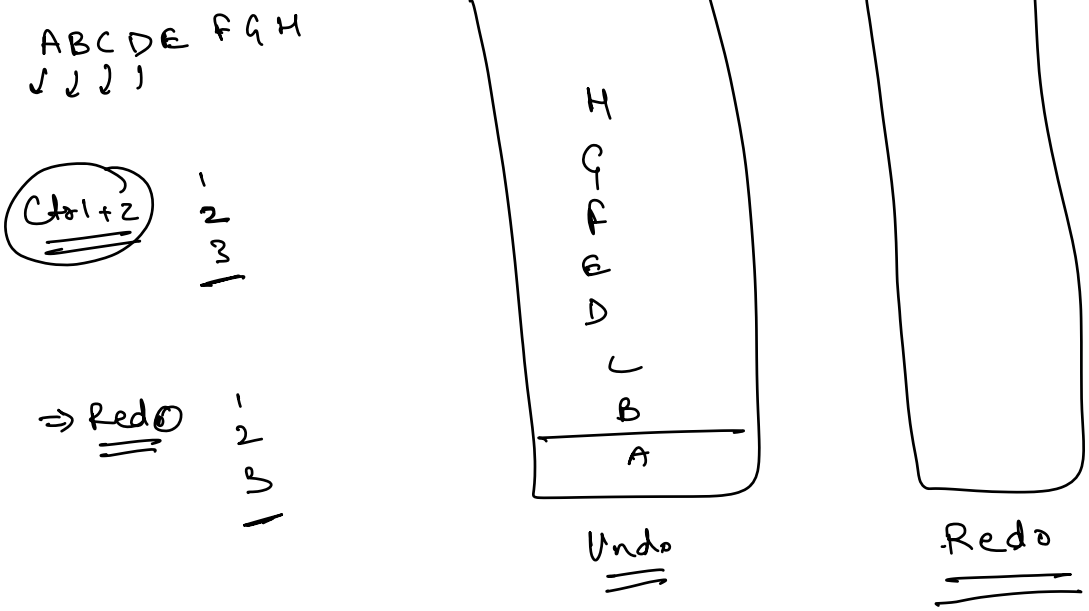
```
main(){
  → print(sum(3));
}
```



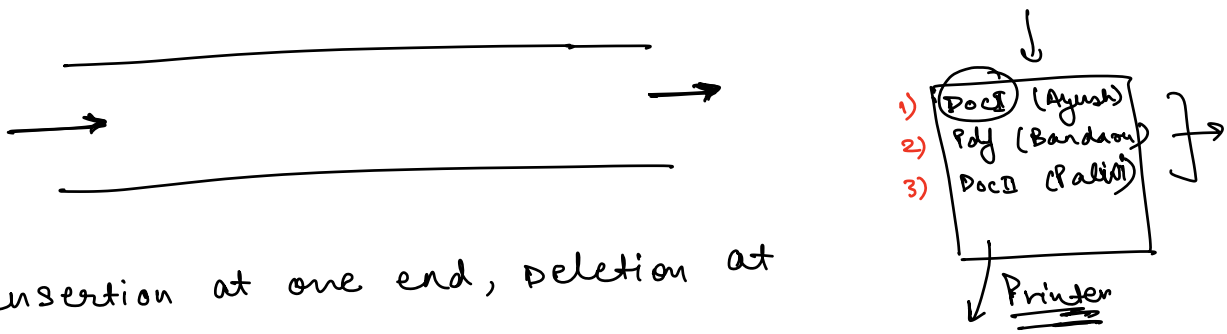


2) Back | forward Button in browser.

3) Undo / Redo operation.



Queue :- (ADT)



* Insertion at one end, deletion at other end.

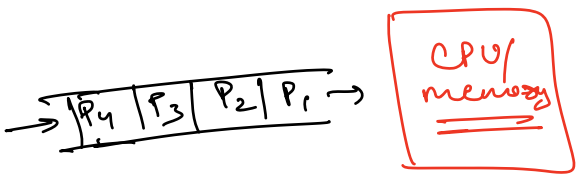
* First In, First Out. (FIFO)

Queue :-
- → Enqueue (push | Insert)
- → Dequeue (Pop | delete)
- → isEmpty()
- → front() | top | peek).

Computer Science applications of Queue :-

1) Kafka | ActiveMQ | RabbitMQ. (HLD classes).
   ↳ Implementation of Queue.

2) Scheduling in OS :-
   ↳



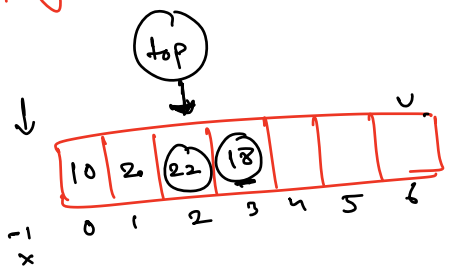
Implementation :-

1) Stack :-
   - ↳ Array ✓
   - ↳ Dynamic Array ←
   - ↳ Linked list. ✓

```
int arr[100];
int top = -1; // Stack is Empty

void push(int data){
    if(top < 99){
        top++;
        arr[top] = data;
    }
}
```

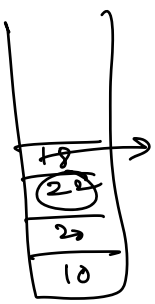
TC: O(1)




```
void pop(){
    if(top > -1)
        top--;
}
```

TC: O(1)

push(10)
push(20)
push(22)
push(18)
pop()



```
int top() {
    if (top > -1)
    return arr[top];

}
```

TC: O(1)

Python :-

A = [] ← List

A.append(x) ← push()

A.pop() -

Java :- Collections.

```
Stack<int> St = new Stack<>();
    St.push(x)
    St.pop();
```

C++ : STL

```
Stack<int> st;
    st.push(x)
    st.pop()
```

Google.com ↓

Queue using Array :-

{ Insertion :- O(N)
  { Deletion : O(1)

HW :- Implement Queue using Array.

Stack implementation using L.L :-

```
Class Node {
    int data;
    Node next;
    Node (int x) {
        this.data = x;
        this.next = NULL;
    }
}
```

```
Class Node:
    def __init__(self, x):
        self.data = x
        self.next = None
```

- push(x) : Add at head ⇒ O(1)
- pop() : Delete at head ⇒ O(1)

Queue using L.L :-

Linked List :-

1) Add at front ⇒ O(1)
2) Add at tail ⇒ O(1)
3) Delete from front ⇒ O(1)
4) Delete from Tail ⇒ O(N)

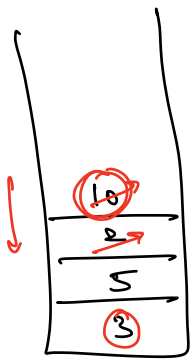
Enque ⇒ Add at tail ⇒ O(1)

Dequeue ⇒ Delete from front ⇒ O(1)

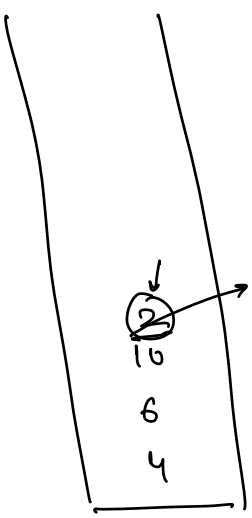
Q Design a stack, which will have getMin() function.
getMin() ⇒ returns the minimum value in stack.

Amazon | GS | MS | Paytm | Uber | Vmware - - -

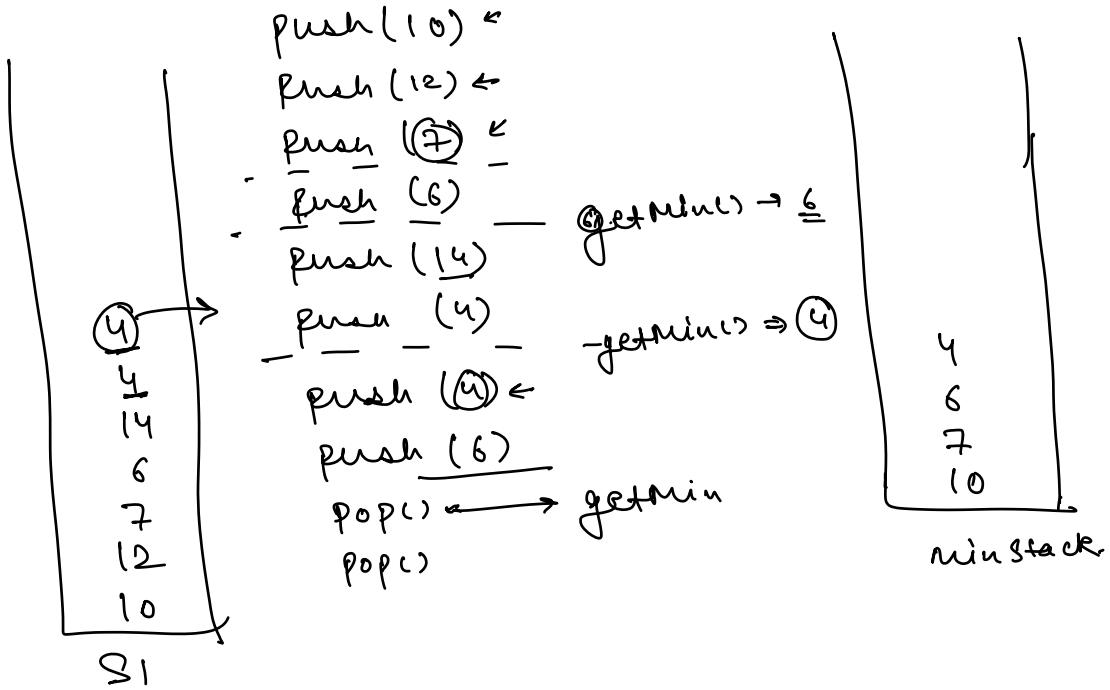
push(3) ✓
push(5) ✓
getMin() → (3) ✓
push(2)
getMin() → (2) ✓
push(10)
getMin() → (2)
pop() ✓
pop() ✓
getMin() → (3)



min = ~~∞~~ ~~4~~ (2)



push(4)
push(6)
getMin() → 4
push(10)
push(2)
getMin() → 2
push(4)
pop()
getMin() → 2
pop()
getMin() → (4)


push(10)
push(12)
push(7)
push(6)
push(14)
push(4)
push(4)
push(6)
pop()
pop()
getMin() → 6
getMin() ⇒ 4
getMin
4
14
6
7
12
10
S1
4
6
7
10
minstack

```
Stack<int> s1; → ADT.

getMin() ← Stack minStack;

Void pushMinStack (x) {          )
    s1.push(x);
    if(minStack.isEmpty() ||
            x ≤ minStack.top() ) {
        minStack.push(x);
    }
}                                ⇒ O(1)

int getMin() {
    return minStack.top();       ⇒ O(1)
}

Void pop() {                     ⇒ O(1)
    temp = s1.top();
    s1.pop();
    if( minStack.top() == temp
        minstack.pop();
}
```

Support @ scaler.com.