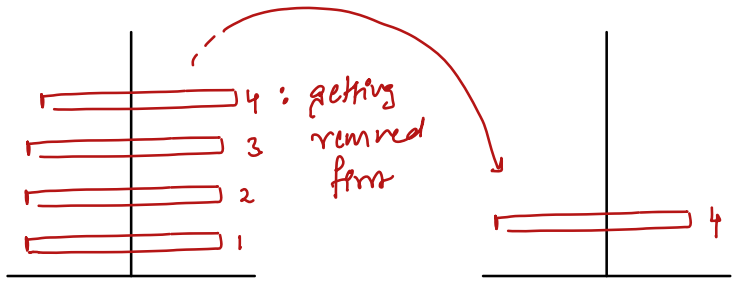
Todays Content

- Stacks Intro & Implementation
- Queue Intro & Implementation

Stacks: Property followed FILO / LIFO → last in first out
↳ first in last out



Stacks : (Real World Use Cases)

- Storing function calls
- Expression Evaluation
  Infix → Postfix → Evaluate Postfix Expressions
- Undo / Redo
- [←, →] prev & next in browser

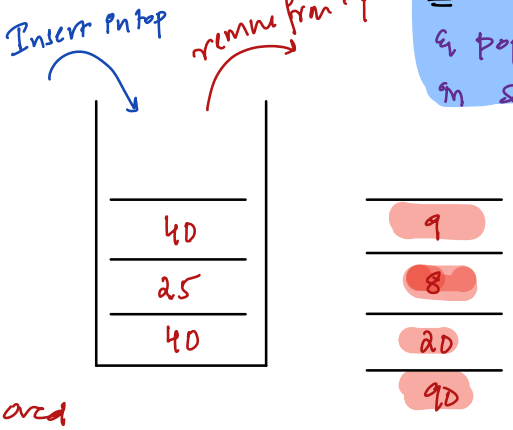
Operations allowed by Stack

TC for each operation in a Stack take O(1)

- push(x) :
- pop() : // We delete element at top
- top() / Peek() : // Return top element
- Size() : // Elements present in stack
- isEmpty() : // Check if stack empty or not

Eg: Operations

- ✓ push(40)
- ✓ push(20)
- ✓ push(8)
- ✓ push(9)
- ✓ pop() : 9 is removed
- ✓ top() : return 8
- ✓ pop() : 8 is removed
- ✓ pop() : 20 is removed
- ✓ top() : 40
- ✓ push(25)
- ✓ push(10)
- ✓ push(90)
- ✓ pop() : 90 is removed
- ✓ top() : return 40



obs: In stack push & pop are performed in same side

obs: When stack is empty & if we perform pop() / top() we should get error.

It is known as underflow

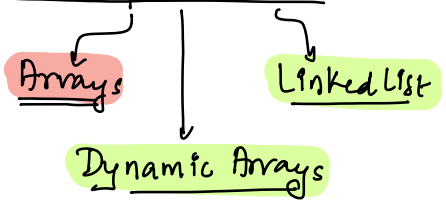
let stack using array[]

ar[5] : | 30 | 40 | 50 | 60 | 70 |

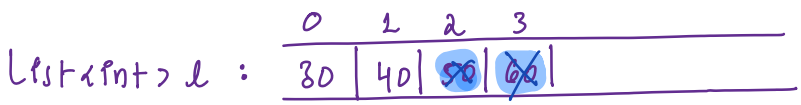
push(30) ✓
push(40) ✓
push(50) ✓
push(60) ✓
push(70) ✓
push(80)

Overflow in stack since we don't have any space to insert new element

Stack implementation



Using Dynamic Arrays:



operations: push(30) ✓ push(40) ✓ push(50) ✓ push(60) ✓ top() 60 pop() 60 deleted pop() 50 deleted

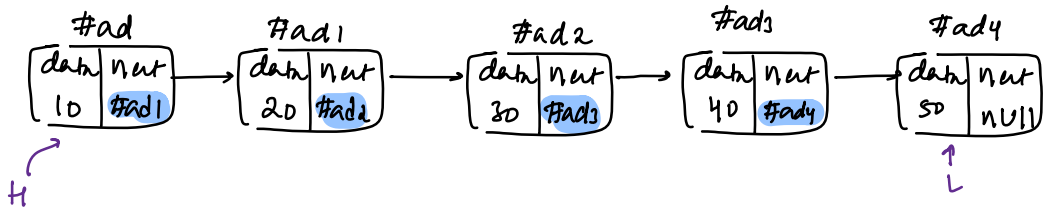
```
void push(n){
    l.insert(n)
}

int top(){
    if(l.size()==0){
        //underflow
        raise error/Exception
    }
    //last element in list
    int n = l.size()
    return l[n-1]
}

void pop(){
    if(l.size()==0){
        //underflow
        raise error/Exception
    }
    //we need to remove last
     index element in list
    l.remove(l.size()-1)
}

int size(){
    return l.size()
}

bool isEmpty(){
    return l.size()==0;
}
```

Stacks Using Linked List

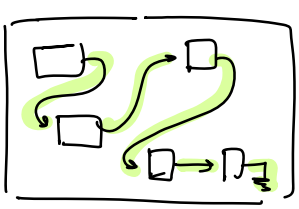
```
class Node{
    int data;
    Node next; // object reference, can hold adares of Node object
    Node(int x){ // Constructor, used to initialize attributes at
        data = x                 time of creating object itself
        next = NULL
    }
}
```

Operations:

push(10), push(20), push(30), push(40), push(50), pop(), pop()



Issue: After deleting last node, we cannot update L, ref to prev node,



Operations in linked list

Insert at back : O(1)

Insert at start : O(1)

Delet at back : O(N)

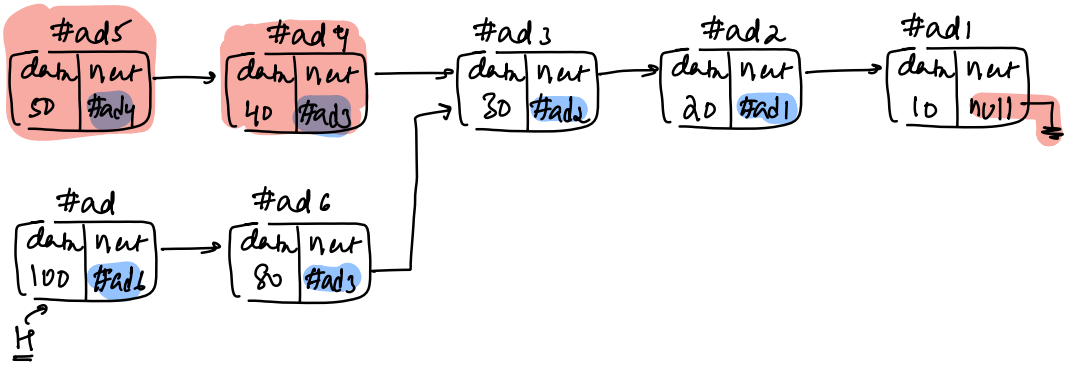
Delet at start : O(1)

} follow these two, 2 perform stack operations

operations:

push(10) push(20) push(30) push(40) push(50) pop() pop() push(80) push(100)

50 40



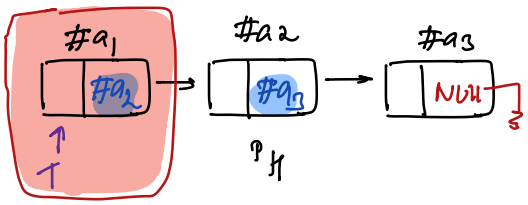

```
class Node{
    int data;
    Node next;
    Node(int x){
        data = x
        next = NULL
    }
}
```

```
Node H = NULL // global variable
int  S = 0
void push(int a){ TC: O(1)
    Node n = new Node(a);
    n.next = H
    H = n;
    S = S+1 // size increased by 1
}
```

```
int top(){
    if( H == NULL){
        // underflow
        raise err/exception
    }
    return h.data
}
```

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

```
int size(){
    return S
}
```




```
Void pop() {
    if( H == NULL) {
        // underflow
        raise err/Exception
    }
    Node t = H
    H = H.next
    deallocate(t) / destructor(t) / free(t)
        // delete node object refered by object reference t
    S = S-1; // reducing size by 1
}
```

Check this in your language of choice, function is it done automatically

Predefined Stack library // Check this library in your language of choice //

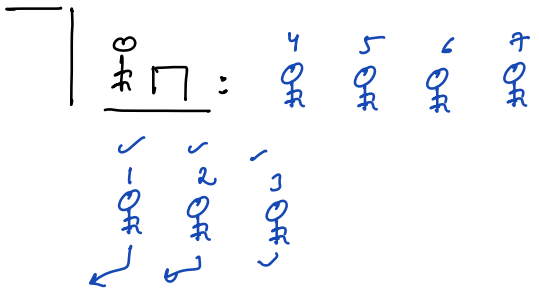
Pseudocode:

```
Stack<int> st // stack, each each is int
st.push(n)
st.pop()
st.top()
st.size()
st.isEmpty()
```

TC of each operation is O(1)

Queue: follow FIFO property or LILO property

- FIFO property → first in first out
- LILO property → last in last out



properties:

enque(x): We insert at rear end

deque(): We delete front element

front(): // First element

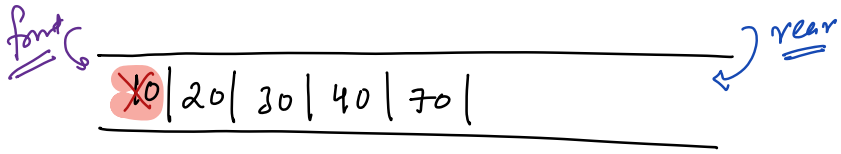
rear(): // Element just entered queue

size():

isEmpty():

Eg1:

en(10) en(20) en(30) en(40) de() fr() en(70) re() fr()
✓ ✓ ✓ ✓ 10 20 ✓ 70 20



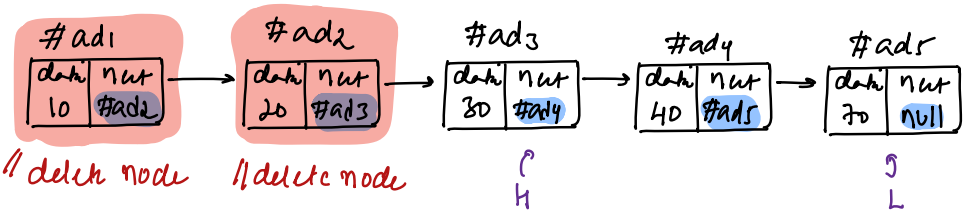
Implementation of Que:

- Arrays — TODO
- LinkedList
  - insertback(): O(1)
  - deletfront(): O(1)
- Dynamic arrays
  - insert at back(): O(1)
  - delet at start(): O(N)

Note: deleting take a lot of time

linkedlist:

en(10) en(20) en(30) en(40) de() → 10 fr() → 20 en(70) re() → 70 de()




```
class Node{
    int data;
    Node nent;
    Node(int x){
        data = x
        nent = NULL
    }
}

Node H = null
Node L = null
Int S = 0
```

```
void enque(int a){
    Node n = new Node(a)
    if(H == NULL){ // 1st node
        H = n, L = n
    }
    else{
        L.nent = n // insert at back
        L = n // update L reference
    }
    S = S+1 // increase size
}
```

```
int front(){
    // 1st element
    if(h == NULL){
        // underflow
        raise error/Exception
    }
    return h.data
}
```

```
int rear(){
    // 1st element
    if(h == NULL){
        // underflow
        raise error/Exception
    }
    return t.data
}
```

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

```
void deque(){
    if(h == NULL){
        // underflow
        raise error/Exception
    }
    Node t = h
    h = h.next
    free(t)
    S = S-1; // reducing size by 1
}
```

```
int size(){
    return s;
}
```

Inbuilt library Queue // Please check in your language

Pseudocode:

queue<int> q; → each element is of int

q.enque() // Insert at rear end

q.deque() // delete at front end

q.front() // Element at front

q.rear() // Element recently inserted

q.size() // Elements in queue