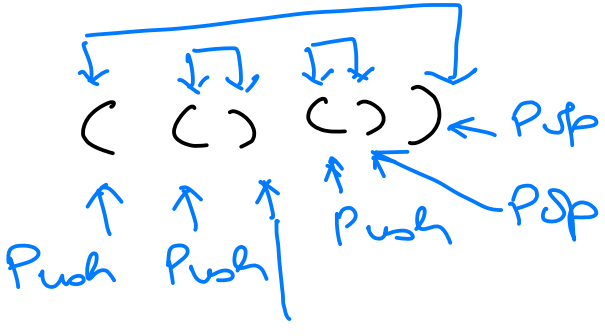
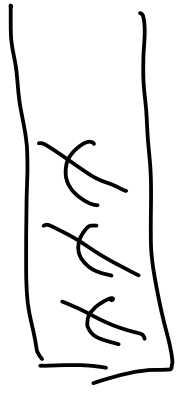
① Check for balanced parenthesis.

( ) ✓)(× (×) ×

((( ))) ✓ (( ) ( )) ✓

(( ) ×





Pop
→ Check if opening parent. poped out is of same type as closing.

→ When I/p is over and stack is empty then Report Success.

{) ×

IsBalanced(expr)

- While expr is not over, get a char from it.
  - if char is '(' then Push char on stack.
    Else // char is ')'
    - if stack is empty then
      - Report failure
      - Stop
    - Pop a value from stack
    - if char is ')' and value is NOT '(' then
      - Report failure
      - Stop
- if stack is not empty then
  - Report failure
  - Stop
- Report success
- Stop

Unit Test → TDD
Test Driven Development

JUnit

(2) Implement queue using Stacks.

↳ Push()
↳ Pop()

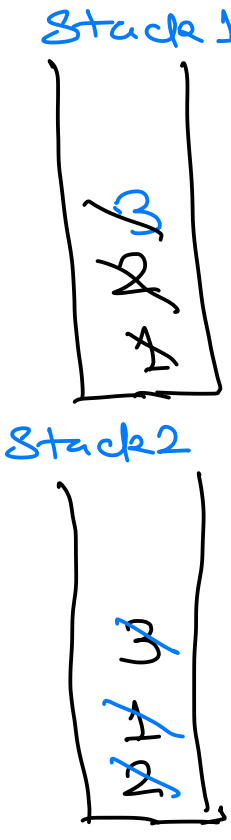
Add Q (1)
Delete Q () ⇒ 1

Add Q (1)
Add Q (2)
Delete Q () ⇒ 1
Delete Q () ⇒ 2

Add Q (1)
Add Q (2)
Delete Q () ⇒ 1
Add Q (3)
Delete Q () ⇒ 2
Delete Q () ⇒ 3

Add Q()
↳ Push()

Delete Q()
↳ Pop()



Add Q(1)
↳ Push(1) in Stack1

Add Q(2)
↳ Push(2) in Stack1

Delete Q() ⇒ 1
→ if Stack2 is empty
→ One by one Pop from Stack1 and Push in stack2.
↳ Pop() from Stack2

Delete Q() ⇒ 2
↳ Pop() from Stack2

Add Q(3)
↳ Push(3) in Stack1

Delete Q() ⇒ 3
↳ if stack2 is empty
}
↳ Pop() from stack2

Add Q (1) → Push (1) in Stack 1

Add Q (2) → Push (2) in Stack 1

Delete Q ()

Add Q (3)

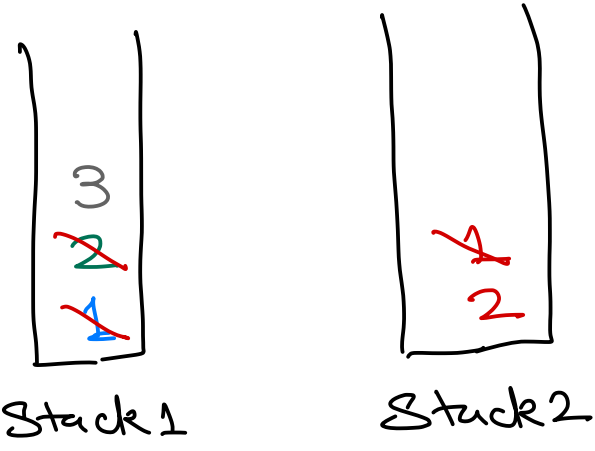
↓

Push (3) in Stack 1

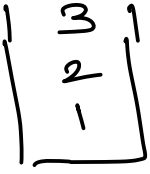
1

→ if stack 2 is empty then
→ One by one pop from stack 1 and push in stack 2

→ Pop from stack 2



Store ⇒ 1 2 3



Get from Stack ⇒ 3 2 1

and Put in another stack

Get from another stack ⇒ 1 2 3

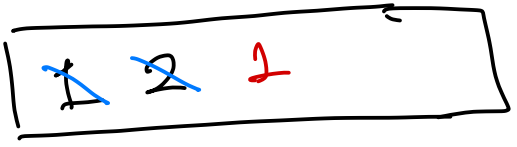
(3) Stack using queues.

Push (1) → Add Q(1) in Queue 1

Push (2) → Add Q(2) in Queue 1

Pop ( ) ⇒ 2

→ Move all elements from Queue 1 to Queue 2, except the last one.



Queue 1

elem → 1 2

Queue 2

→ Last element removed from queue 1 is result

→ Put back all elements from queue 2 to queue 1

→ Return result

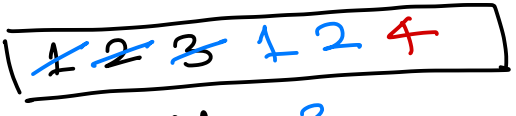
Push (1) → Add Q(1)

Push (2) → Add Q(2)

Push (3) → Add Q(3)

Pop() → 3

Push (4) → Add Q(4)

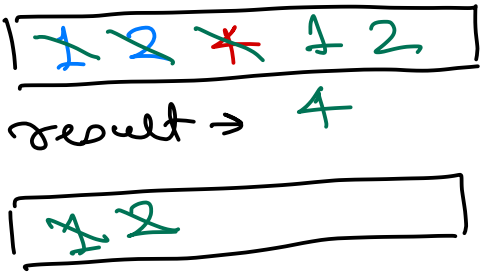


Queue 1

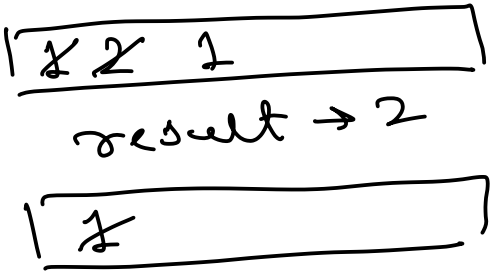
result → 3

Queue 2

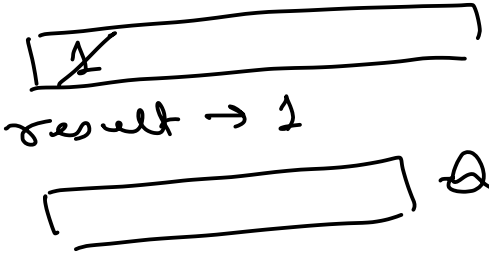
pop() → 4



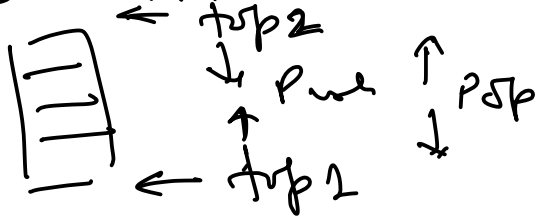
pop() → 2



pop() → 1



④ Implement two stacks



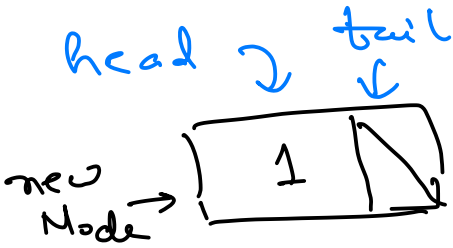
Push(stackno, elem)

Linked List

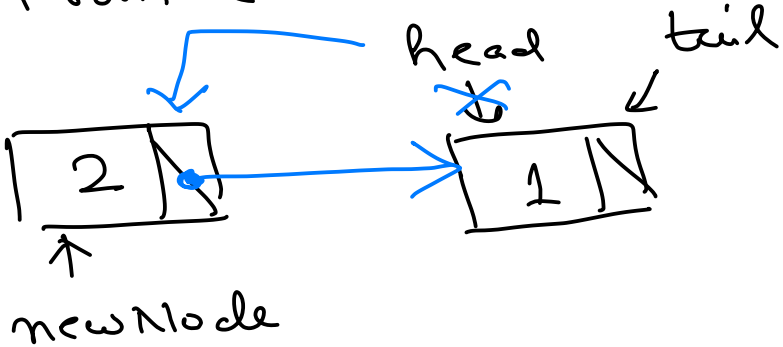
Create a linked list

- Add new element at front
- Add new element at end of list
- Insert in a list ⇒ Creating Sorted list.

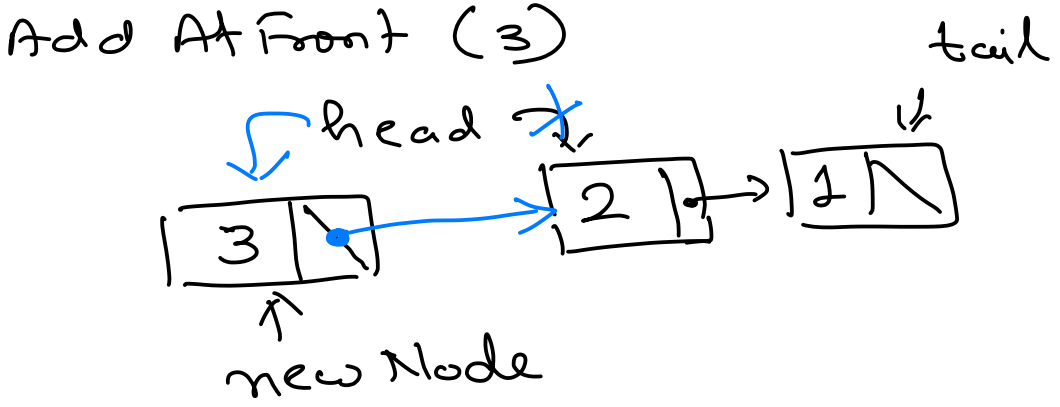
Add At Front (1)

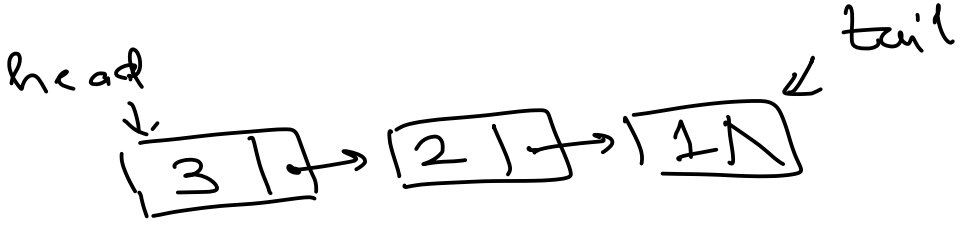


Add At Front (2)



AddAtFront(element)
- Make space for new elements, say newNode.
- Store element in newNode's data.
- Set newNode's next to empty.
- if list is empty then
  - Set head and tail to newNode.
  - Stop.
- Set newNode's next to head.
- Set head to newNode.
- Stop.





Dynamically allocate memory for Node.

```
Class Node {
public int data;
public Node next;
}

Node head;
Node tail;
```