LINKED LISTS

→ Each node contains

- Data
- A pointer to next node in sequence

**** Current != null vs Current.next != null ****

→ Current != null: when you want to traverse the entire linked list, including the last node (start → end)

→ current.next != null: when you need to stop right before the last node (when inserting at end or removing last node)

**** Remember ****

→ Update head

→ maintain references

→ advance current pointer

→ Don't access current.next when current is null

SLL

→ To add a node to empty list

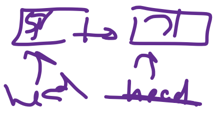
1) create a new node containg data
2) set that newNode's next to current head
3) set head to newNode

→ To iterate through SLL

```
1) create current node
2) current = head
3) while (current != null){
       ...
       current = current.next;
   }
```

→ Operations on SLL

1) addToFront() – O(1)



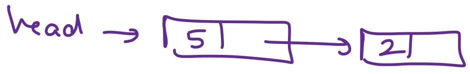
- newNode's next to head
- newNode to head (head = newN)

~~newNode = head~~ ~~head~~

2) addToBack() – O(n)

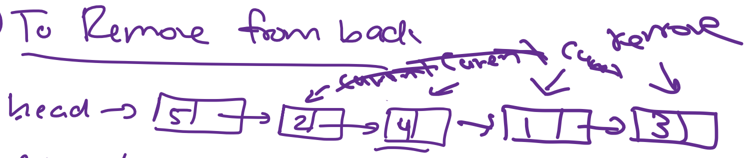
- Iterate until current's next is null NOT until current is null
- General Case:
    1) current node = head
    3) if next is null, you are sitting at the last node & all you need to do is set the next pointer to new node
- Edge Case: if head is null, point head to newNode

3) To Remove from Front



- Save data from head for returning
- head = head.next

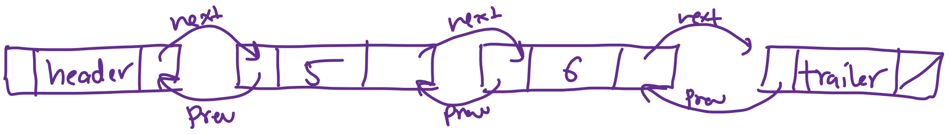
4) To Remove from back



- current = head
- while (current.next.next != null) {current = current.next}

current.next = null;

Doubly LL



- Size : 0 → both head & tail point to null
- Size : 1 → both point to single node

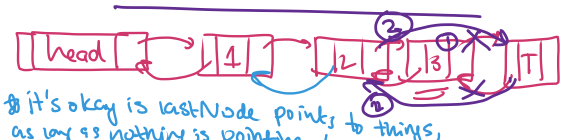
→ To add to front

1) create newNode
2) newNode.next = head
3) head.prev = newNode
4) head = newNode

→ To add to back

1) newNode.prev = trailer.prev
2) trailer.prev.next = newNode
3) trailer.prev = newNode
4) newNode.next = trailer

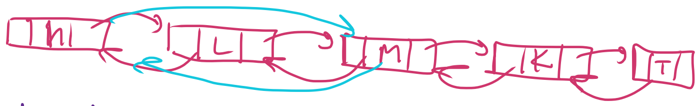
→ To remove from back



* it's okay is lastNode points to things, as long as nothing is pointing to it *

1) trailer.prev = trailer.prev.prev
2) trailer.prev.next = trailer

→ To remove from front



head = head.next

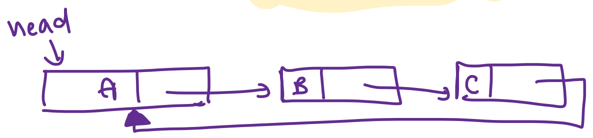
head.prev = null

Circular Linked Lists

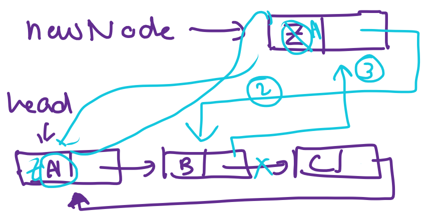
→ last node points back to head

→ can no longer use current == null to check if we've reached end of our list

→ must use current == head to terminate our loop



→ Adding to the front in O(1)



1) create a new empty node
2) newNode.next = head.next
3) head.next = newNode
4) newNodeData = newNode.data
5) newNode.data = head.data
6) head.data = newNodeData

* Don't do:
create a new node, point to head & move head then reset last node to point to new head cuz O(n) *

→ Adding to back in O(1)

1) all steps performed to add to front
2) head = head.next

Stacks

→ push(e) - adds element e to top of stack

→ pop() - removes + returns top element

→ top() - returns not removes the top element of stack
  → (peek())

→ Allows elements stored in the stack to belong to any object type <E>

→ Array-based Stack

- all operations are O(1)
- Stack <Integer> S = new ArrayStack < >();