Linked List

* Limitations of arrays:-

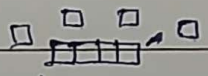
1. Size of array is fixed at time of creation, if you need more space than you use arraylist List & create a new larger array & copy the data over i.e. time-consuming
whereas, Linked list are dynamic can grow or shrink in size as needed.

2. Time complexity of Arrays: O(n).
" " " Linkedlist: O(1).

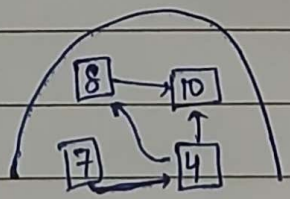
3. As arrays have conts. memory allocation it is reserved regardless of actual usage this can lead to wasted memory if not fully utilized.
Whereas, memory is allocated as needed in L.L.

4. As point (1), to increase the size of array, a new array must be created (double size) & existing elements must be copied over, i.e. O(n) operation.
& earlier one are removed

* Working of a linked list :-



Here all the values or (boxes as in of arrays) are connected each other with a pointer (not pointers of C language) but arrows reflecting reference variables (say), they do not have any indices they are randomly stored (but connected) in Heap memory.



If these boxes are arranged, normal representation of linked list:-

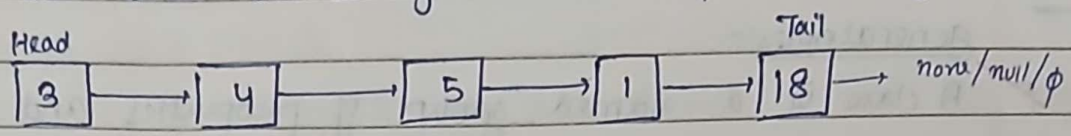
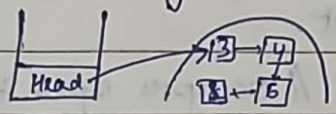


fig: single linked list

Every single item knows about the next item only

e.g.(5 has a crush on 1 but ~~not~~ 1 does not know that similarly 3 has a crush on 4 but 4 does not know ~~by~~ others).

Head is a reference variable that points to the first node.



and vice versa for tail.

Here we cannot get individual values as if we were in arraylist by list.get (5):

They are pointing to the other variable by:-

```
Class Node {
        int val ;
        Node next;
}
```

every node knows which node I direct to & what's my own value.

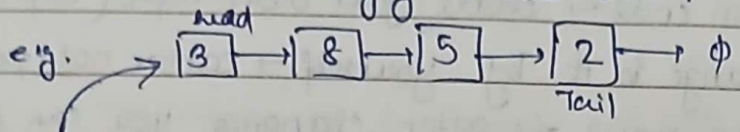
Base - public class LL {
private class Node {
priv. Node next;
" int value;
constr. of value
constr. of both

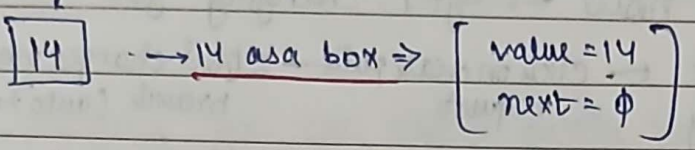
then private Node = head;
" " tail;
" int size;
constr. of size to assign it as zero first

28 06 24

Linked List Continued

- How to insert in singly linked list?

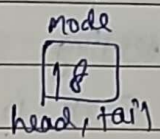




To insert at head:- (after making classes & stuffs)

- node.next = head
- head = node (assign node to head) → ∵ first hogya ye ab
- if (tail == null) → if only 1 node i.e. head = tail
- tail = head
- size++

No loop is running ∴ O(1)



Code ⇒

```
public void insertFirst(int val) {
    Node node = new Node(val);
    node.next = head;
    head = node;
    if (tail == null) {
        tail = head;
    }
    size++
```

node object →

How to display?

```
while (head != ϕ)
    print(head.value)
    head = head.next
```

[wrong] when head reach is null it gives null (else).

rather make a temp. i.e. when head = null ~~how to get null~~ head != null

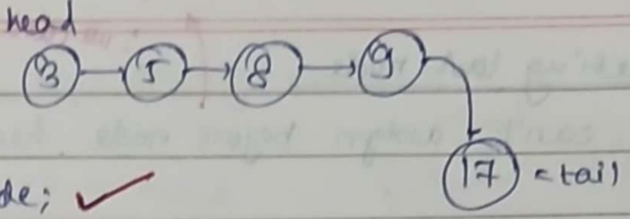
head = temp ← then do this same & move temp

⇒ Check CODE

(3) SUMMARY - 1. in insertFirst, creating the node first is necessary bcoz it forms new head of list + immediately linked to existing list (if any).
2. in insertLast, creating node first is not necessary bcoz you do not need it if list is empty.

Insert at last :-

if like this ⇒



← tail   making ~~inserting~~ at tail

∴ tail.next = node; ✓

O(1) { tail = node ✓   assigning node as tail.

size ++ ✓

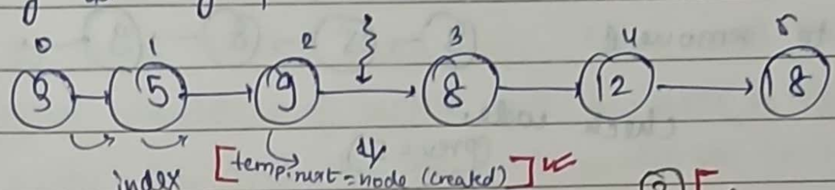
if (tail = null) {
  insertFirst (val) } ;

~~Base condition always first rest after in else.~~
~~∴ just creating to put in tail~~

★ !! ① [since, socho putting content before if statement will get unnecessary if if the IF statement gets true ✓]

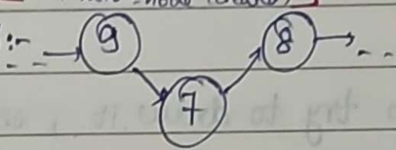
→ check CODE

Inserting at any Index/ position :-



[temp.next = node (created)] ✓

To insert at ~~node~~ 3 :-



② [in insertLast, you don't need to create the node first coz if list is empty you delegate the work to insertFirst. ∴ By creating node early, when you ~~know the list is not empty~~ is unnecessary work.]

✓ ∴ before 3rd index point 2nd index to new node.
⇒ to break 9→8 + store 7 use temp.
store it beforehand (earlier by making insert method).
⇒ Check code ★

∴ not in case for deleting last.

Deleting first node

∵ in linked list, only it knows next one   ~~value~~ = head value
head = head.next;

↳ [Just assign head to its next node] That's It!
→ ∵ head ab agle wale pe chala gya hai so pichla wala aise kbhi access nhi kr skta ∴ removed.

(if only 1 item → head = tail
∴ [head = head.next]
∴ head ⇒ φ
∴ make tail also φ

after taking the value of head ∴ [head = head.next.] ✓
✓ Dont forget to reduce the size ✓

Check code.

Deleting last node

∵ in single LL → it only knows next one not previous one.
∴ we can't assign tail to prev. one directly.

we can't assign before node here (check reason on last page)

O(n) ↓ complexity

1. at (size − 2) assign it to tail & put tail.next = ϕ

we first need value/reference of that node

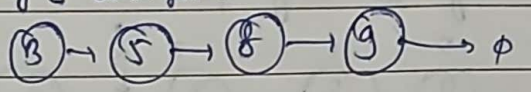
check code

total no. of nodes = n

O(n) ∵ traversing (moving forward)

2. Need to remove at index
   - ↳ go to that index − 1.
   - ↳ then next of 5 assign to 9

eg. to remove 8



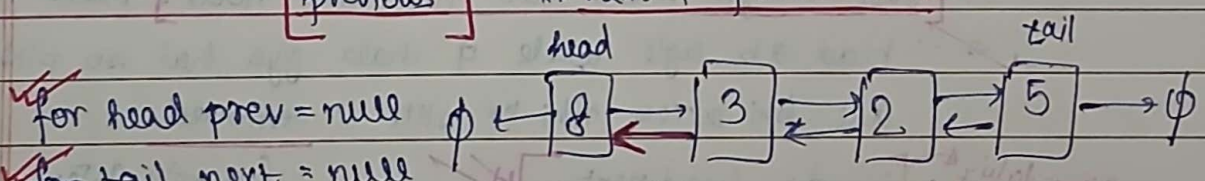
check code.
(prev = 5)

Tip:- always try to draw it, see the approach of ques. then code it.
It's same like watching a JEE ques. & directly using pen without understanding.

* Doubly Linked list

↳ same thing just one single additional reference variable added.

[previous] ∴ next & previous.



for head prev = null
for tail next = null

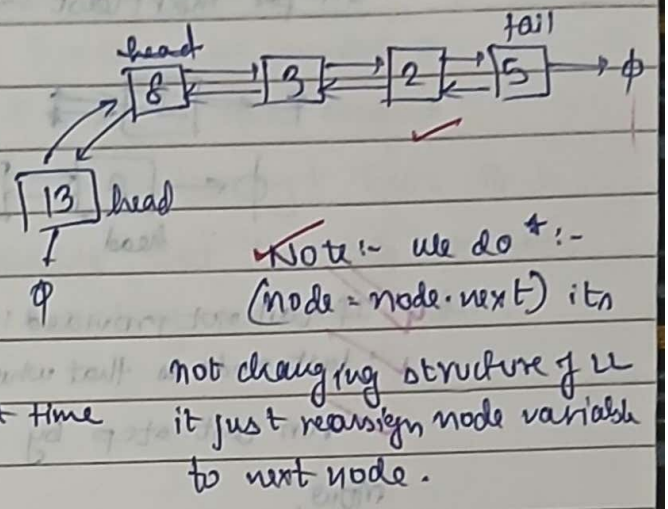
Hence, every single node will have this feature like in single LL.

```
class Node {
    int value;
    Node next;
    Node prev;
}
```

tail is not declared in DLL.

Date : / /
Page :

Insert at first in doubly LL.



⇒ ① node.next = head

② node.prev = null

③ head.prev = node

④ head = node

checking kar sirf agar head hi nhi hua check for null pointer exception like we did in all cases otherwise error

O(1) = constant time

Note:- we do *:- (node = node.next) its not changing structure of LL it just reassign node variable to next node.

★ Note:- all this basic stuff will be simpler & carry forward basically so solve FAANG level Ques. from Kunal Kushwaha videos (after completion of certain topic he adds it next to it.) To make utilize this and to think how to use in industry based purposes.

Note:- This is nothing main thing will come when we do Ques. on it.

Note:- If you memorize you will forget. learn how to think → you will never forget.

+ look for corner cases

Display in reverse

⇒ make a variable last & reverse it from back i.e. last = node with node = node.next till last = ϕ i.e. head se pehle wala null.

⇒ in display() { add → Node last = null ; & inside loop put last = node.

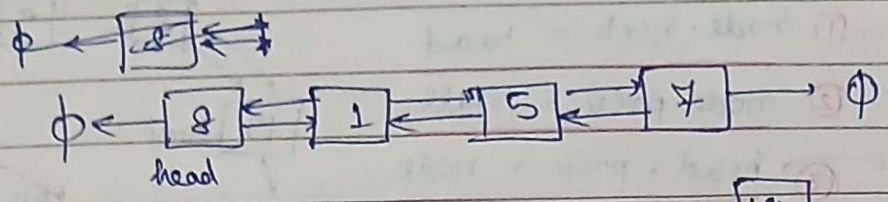
⇒ till last ≠ null (print its value & do last = last.prev)

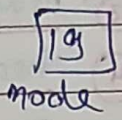
check code

Invert last

↳ Draw on pen & paper first.

Soln for insertLast :-





If tail not provided :-
- last node is that whose next element = null
- run last step by step till it reaches null

Now,
- node.next = null (∵ 19) } very simple
- last.next = node
- node.prev = last
- last & p node pointer will get removed when fn. is over.

→ for exception cases :-

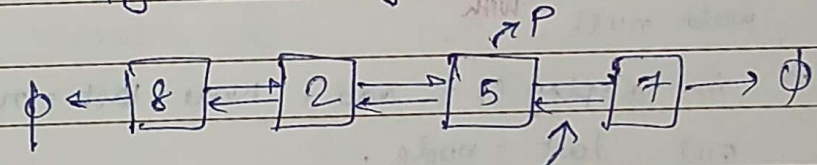
```
if (head == null)
    head = node
    node.prev = null
```

→ check code!

Insert at index

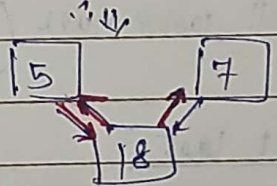
M-1 : same copy as in single LL. (with caution)

M-2 :-



let name of node at just before index be 'p' (here p ⇒ 5)

5→7 ist tha wo ab 18→7 ho gya



1. node.next = p.next [18]→[7]
2. p.next = node [5]→[18]
3. node.prev = p [5]←[18]
4. node.next.prev = node [18]←[7]

VV. easy no thought process just implementation

↳ now check null pointer exception
↳ if you will not check you will get error

Traversing n times to find p ∴ O(n).

check → is node.next gives null pointer? → NO ∵ we created it in mid
is p.next " " " ? → NO ∵ we founded p
is node.prev " " " ? → NO ∵ node created
is node.next.prev = node gives " " ? → Yes it may! bcoz it is possible that the node we are inserting is inserting at last indix.

[this may be null]

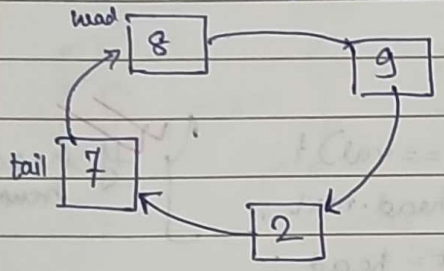
∴ print it only if it's not null

check code.

- Deletion in DLL ★ Hint → think on your own.

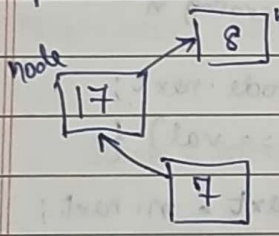
✱ Circular Linked List

↳ structure similar we have in singly linked list + here no thing as null.




```
class Node {
    int value;
    Node next;
}
```

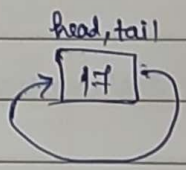
To insert a value in b/w head & tail:-



tail.next = node
node.next = head
tail = node

null pointer exception → head & tail are null
i.e. no items in LL

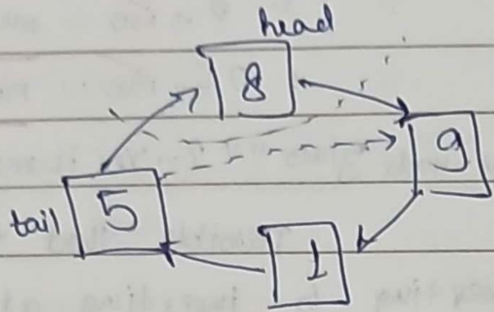
```
if (head == null) {
    head = node
    tail = node
}
```



O(n) How to display? start from head while again node != head
∴ print once & keep going till end ⇒ DO WHILE loop exist atleast once.
check code!

here traverse → मतलब node, ya temp yea else को LinkedList mein chalana upto derived destination.

- To delete something :-



Head
if head only { value = 8
∴ head = head.next
tail.next = head

random
if to delete → value = 1

O(n) {
- Start node from head then check via node.next to whom to delete.
- If to delete one found then node.next = end.next
- if returned back to head then its value does not exist.

There's no as such null pointer exception in case of deletion bcoz →

```
if (node == null)
    return;
```

↓
nothing to delete then.

```
If (node.val == val) {
    head = head.next;
    tail.next = head;
    return;
```

} If head to remove

If some value not head to delete ⇒

```
do {            traversing n
    Node n = node.next;
    if (n.val == val) {
        node.next = n.next;
    } break;
    node = node.next;
} while (node != head);
```

↓
till it reaches head again