#Linked List

* Limitations of arrays:-

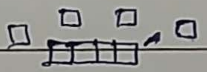
✓ Size of array is fixed at time of creation, if you need more space then you use arraylist list & create a new larger array + copy the data over i.e. time-consuming
whereas, Linked list are dynamic can grow or shrink in size as needed.

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

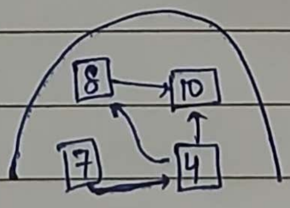
✓ As arrays have conts. memory allocation it is reserved regardless of actual usage this can lead to wasted memory if not fully utilized.
Whereas, memory is allocated as needed in L.L.

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

* 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 represtation of linked list:-

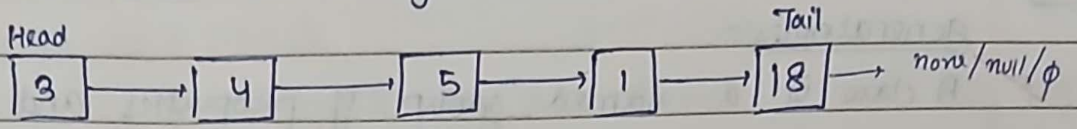
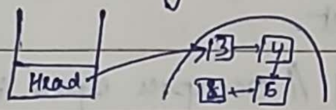


fig: single linked list

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 aly 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.

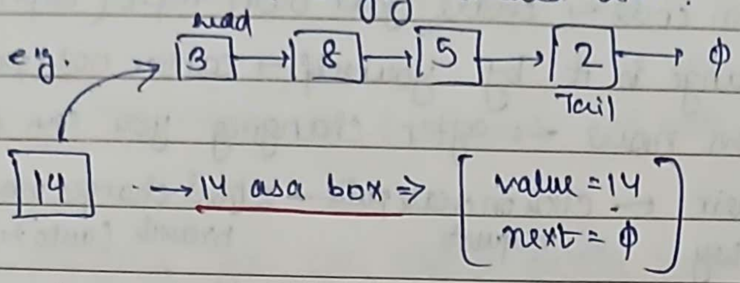
Basics:- public class LL {
private class Node {
priv. Node next;
" int value;
}
constr. of value
} constr. of both

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

Date 28 06 24
Page:

#Linked List Continued

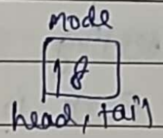
• How to insert in singly linked list?



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

```
✓ node.next = head
✓ head = node      (assign node to head) ∵ first hoggya ye ab
✓ if (tail == null)  } → if only 1 node i.e. head = tail
✓     tail = head    }
✓ size++
```

no loop is running ∴ $\boxed{O(1)}$



code ⇒

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

✓ How to display?

```
while ( head != ϕ)          }  [wrong] when head reach is null
      print (head.value)    }  it gives null (else).
      head = head.next      }  rather make a temp.
```

i.e. when head = null ~~how to get null~~ head!=null

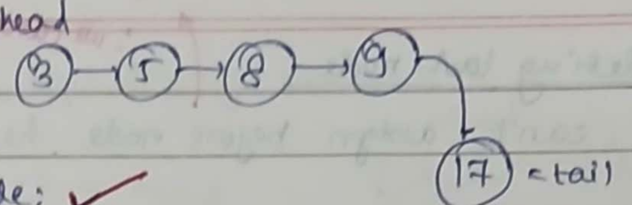
[head = temp] ✓ then do this same & move temp

⇒ Check CODE

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

Insert at last :-

• if like this ⇒



making ~~inserting~~ at tail

```
∴ tail.next = node;  ✓
  tail = node  ✓      assigning node as tail.
  size ++  ✓
  if (tail = null) {
      insertFirst (val) };
```

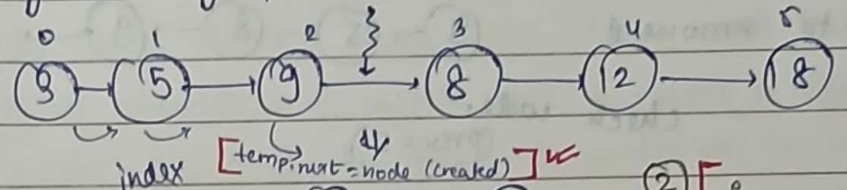
O(1)

~~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 ✓]

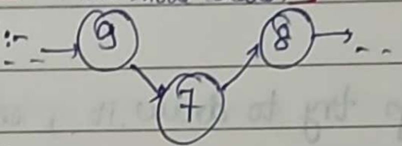
→ check CODE

Inserting at any Index/position :-



[temp.next = node (created)] ✓

To insert at ~~node~~ index 3 :-



✓∴ before 3rd index point 2nd index to new node.
⇒ to break 9→8 & store 7 use temp.
store it beforehand (carrer by making insert method), or else
⇒ check code ☆

(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.]

Deleting first node

∵ In linked list, only it knows next one → ∴ not in case for deleting last.

int value = head.value
head = head.next;

↳ [Just assign head to its next node.] That's it!
→ ∵ head ab agle wale p chala gya hai so pichle wala aise bhi access nhi kr skte ∴ removed.

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

(if only 1 item → head = tail
∴ $\boxed{\text{head = head.next}}$
∴ head ⇒ ϕ
∴ make tail also ϕ

Check code.

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

Deleting last node

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

O(n) + complexity

1 ⇒ at $\underline{(size - 2)}$ $\underline{\text{assign it to tail}}$ + put $\boxed{tail.next = \phi}$

↓ we first need value/reference of that node

check code

total no. of nodes " n ↑ O(n) ∵ traversing (moving forward)

• now to remove at index
↳ go to that index − 1.
↳ then next of 5 assign to 9

eg. to remove 8

$$\begin{tikzcd} 3 \arrow[r] & 5 \arrow[r] & 8 \arrow[r] & 9 \arrow[r] & \phi \end{tikzcd}$$

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.
↓
$\boxed{\text{previous}}$ ∴ next + previous.

for head prev = null
for tail next = null

$$\begin{tikzcd} \phi & \overset{head}{8} \arrow[l] \arrow[r, shift left] & 3 \arrow[l, shift left] \arrow[r, shift left] & 2 \arrow[l, shift left] \arrow[r, shift left] & \overset{tail}{5} \arrow[l, shift left] \arrow[r] & \phi \end{tikzcd}$$

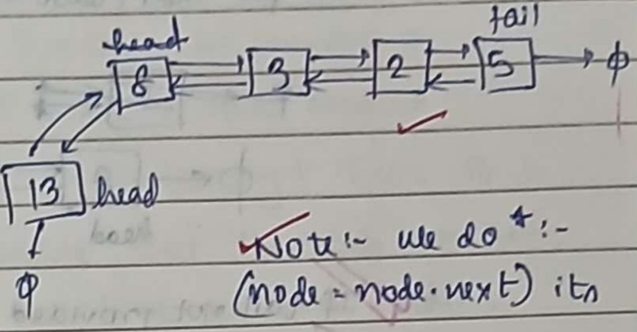
Hence, ~~ev~~ every single node will have this feature like in single LL. →

```
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

$\underbrace{\qquad\qquad}$ O(1) = constant time

checking here since agar head hi nhi hua

check for null pointer exception like we did in all cases otherwise error

~~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 topics 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 → + look for corner cases you will never forget.

Display in reverse

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

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

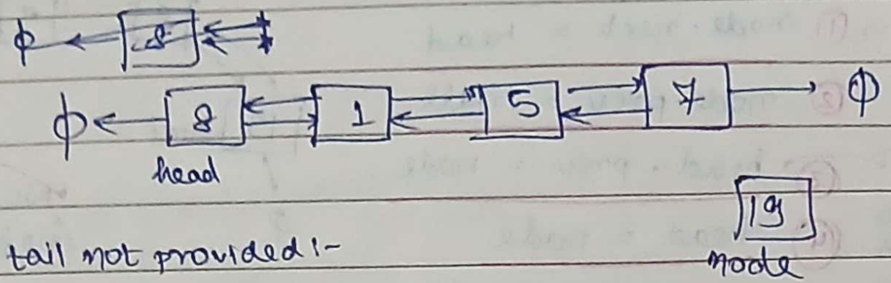
⇒ till last ≠ null (print its value & do last = last.prev)

HW

Insert last

↳ Draw on pen & paper first.

<u>soln for insertlast</u> :-



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)
- last.next = node } very simple
- node.prev = last

7 → 9

- last & ~~p~~ node pointer will get removed when fn. is over.

→ for exception cases:-

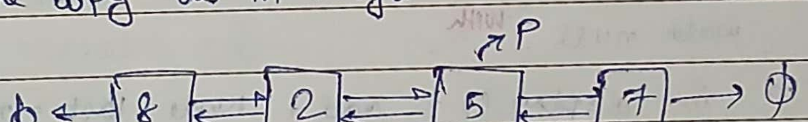
```
if (head == null)
    head = node
    node.prev = null
```

→ check code!

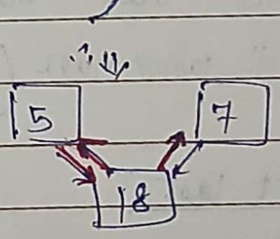
Insert at index

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 tha wo ab 18 → 7 ho gaya

1. node.next = p.next $\boxed{18}\rightarrow\boxed{7}$
2. p.next = node $\boxed{5}\rightarrow\boxed{18}$
3. node.prev = p $\boxed{5}\leftarrow\boxed{18}$
4. node.next.prev = node $\boxed{18}\leftarrow\boxed{7}$

v.v. easy no thought process just implementation

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

Date : / /
Page :

↙ prev. page

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 $\underbrace{\text{node.next}}$.prev = node gives " " ? → Yes it may! bcoz it is possible that the node we are inserting is inserting at last index.

[this may be null]

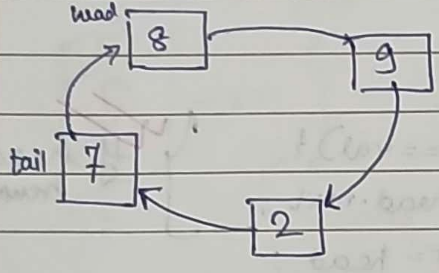
∴ print it only if it's not null

check code.

- Deletion in DLL ★ H.W → think on your own.

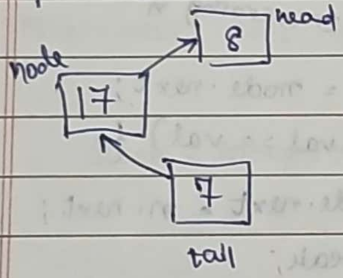
* Circular Linked List.

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



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

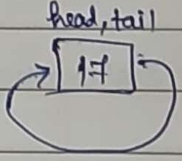
- To insert a value in b/w head & tail :-



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

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

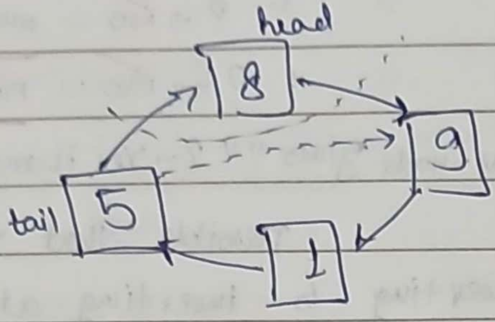
```
if (head == null) {
    head = node
    tail = node
}
```



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 ya else को LinkedList mein chalana upto desired destination.

To delete something :-



Head

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

random

if to delete → value = 1

- 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.

$O(n)$

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