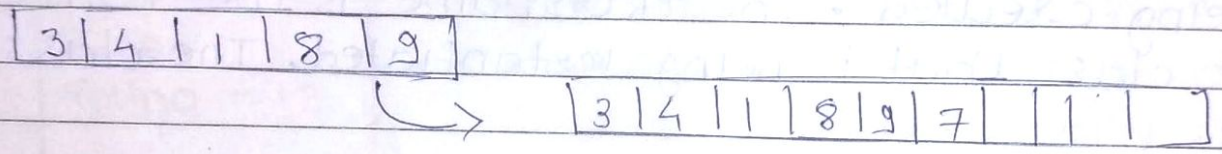
* Linked List *

Page No.
Date :

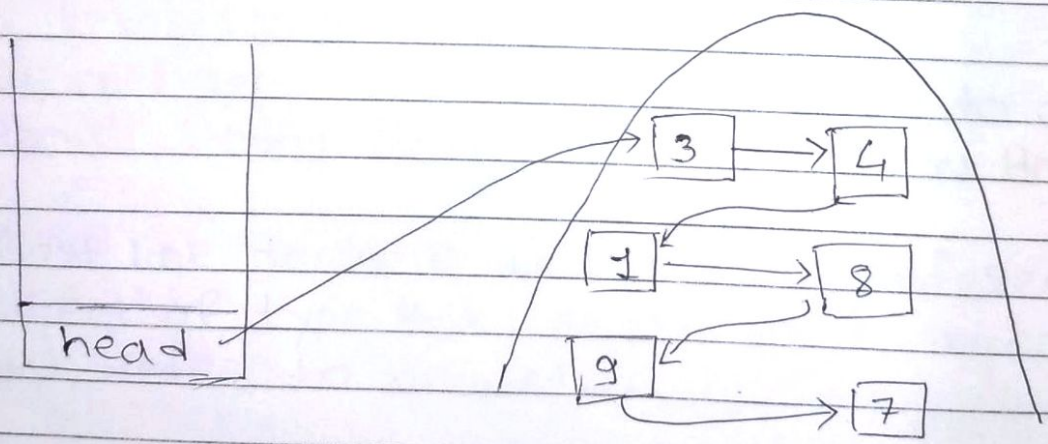
* Limitations :-

1) We can not add element in the array beyond the size of array.

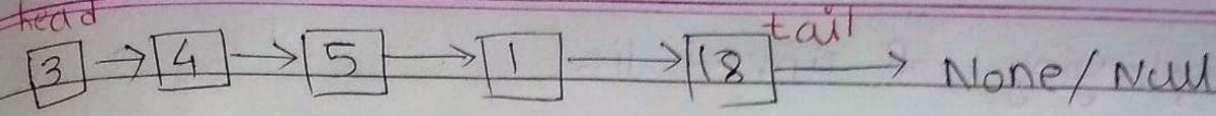
2) If we create arraylist and add new elements to it, it will copy all those elements and creates a new list even though it is of constant time O(1), but it's limiting us.



instead of doing this we can break this into different boxes.



Single linked list —
This is basically the linked list, now items are not like continuous memory allocation, so here items are put in different memory (not continuous) & they are connected through pointer.



Here,
- every box is called as "node"
- Head - it is a reference variable that points to very first node
- Tail - It is a reference variable that points to last node

*So this each node can have the following properties,

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

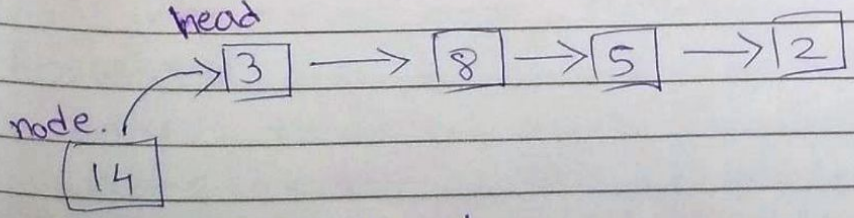
* Each node don't have an idea about how many nodes are there in the linked list, each node only have the idea about what it's value & whom it is pointing.

* steps to solve problem:-

Ex - insert an element at 1st position.

Step1 :- visiualize what to do.

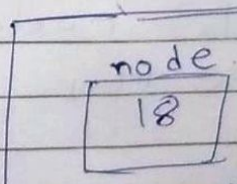
Step2 :- How to do it.



node.next = head

node = head

if (tail == null)
tail = head



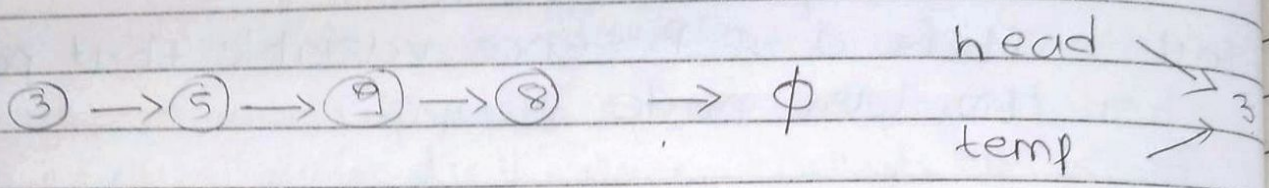
head, tail

Page No.
Date :

* Displaying of linked list.

wrong

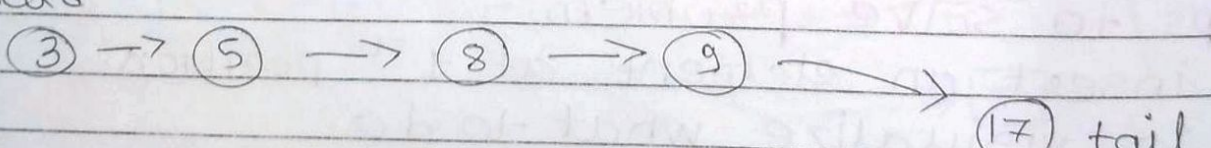
```
while (head != ϕ)
    print (head.val)
    head = head.next.
```



- If we are moving our head it will change structure of our linked list, because head always needs to be point to the 1st node. So we took a temp. which is also pointing to head and we ~~wil~~ this temp will keep moving forward till it reaches end.

```
temp = head. // it basically pointing to
                the same object which
                head is pointing to
```

* For inserting value at end.

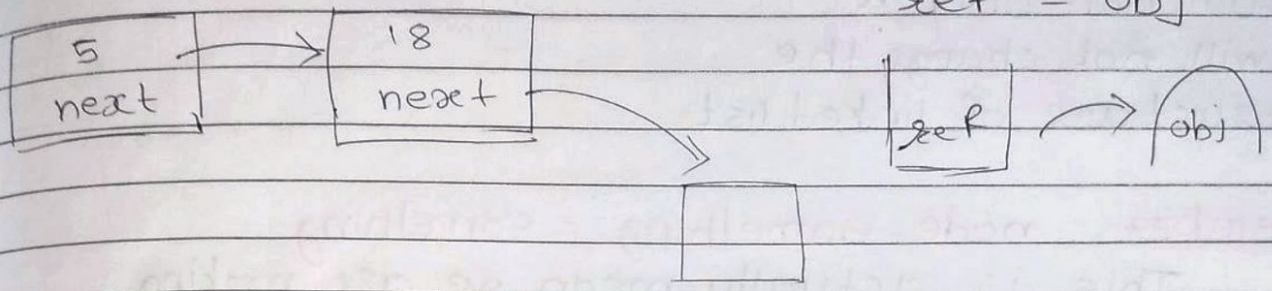



```
if (tail = null) {insertfirst (val)};
tail.next = node.
tail = node;
size ++;
```

Page No.
Date :

* All about next :-

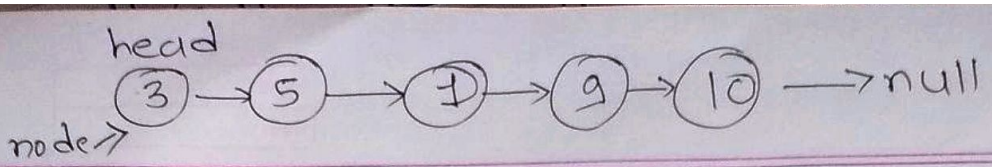
~~In~~ Each node is a data structure, internally it will look like this



- These nodes are not index values, which we can point directly, each node will be some where in the memory
- If we want to go from one node to next node we can say nodeName.next
- In case of single linked list, there is no such a variable which will put us behind the node.
- next is just a reference variable of type node pointing to something that is node, that's why it's pointing to type node.

* All about head :-

whenever we traverse, whenever we move forward we don't change the head, we say node = head.
here node is the temporary variable.



Page No.
Date :

node = head.

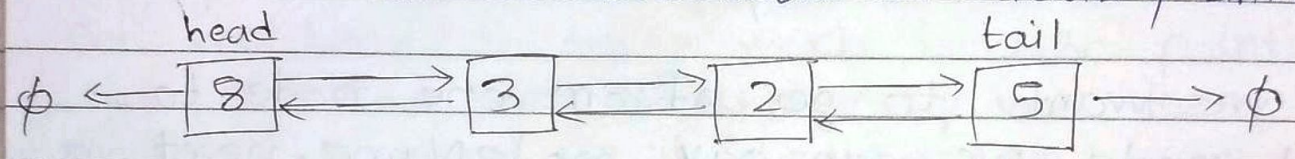
it's scope will be in function only & it will not change the structure of linked list

Remember :- node.something = something

This is actually mean we are making a change.

Doubly LL :-

Basic structure of doubly linked list-

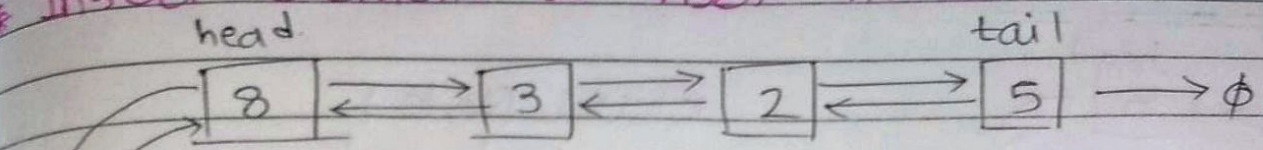


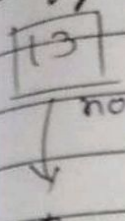
Each node will have this structure -

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

Date :

* Insert element at first index.

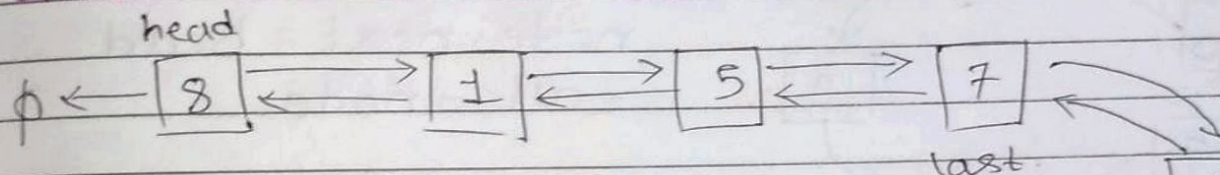


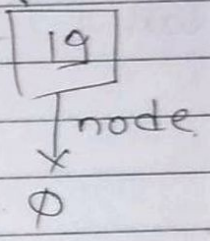


node.next = head
node.prev = ϕ
head.prev = node
head = node

Check for null pointer.

* Insert element at last index.

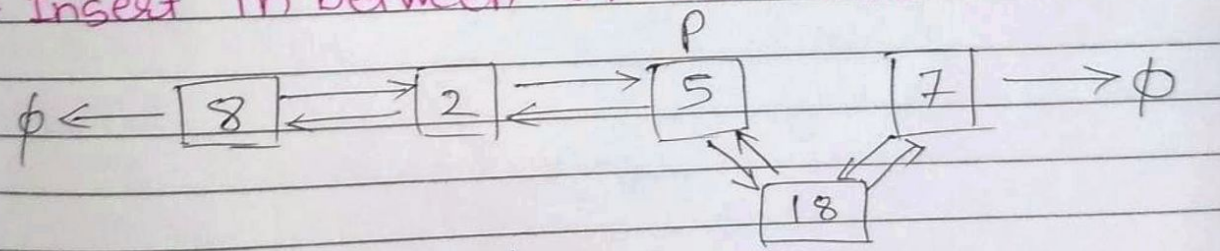




node.next = ϕ
last.next = node
node.prev = last

if (head = ϕ)
node.prev = ϕ
head = node

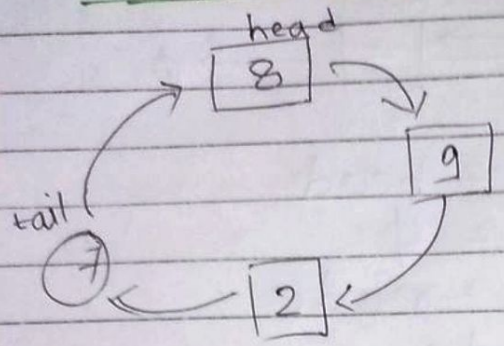
* Insert in between of the list.



node.next = p.next
p.next = node
node.prev = p
node.next.prev = node

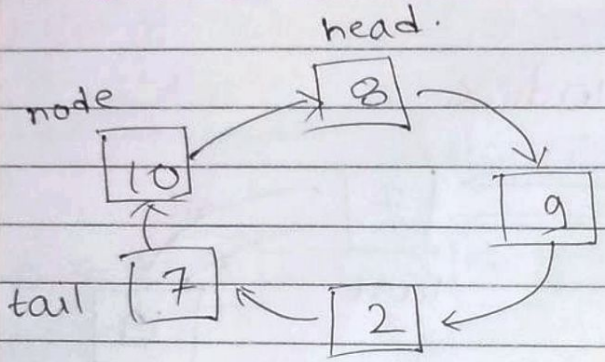
Page No.
Date :

* Circular LL :-




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

* Insert element




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

h, t
17

```
if (head == ϕ)
    head = node
    tail = node
```