Welcome ☺

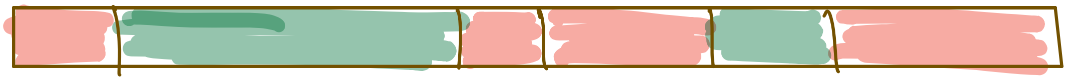
Advance Module 3 → LL
Stack / Queues
Trees
DP
Graphs
} **

Agenda: Linked List
Opera^ns
2-3 questions

Linked List

Arrays ⇒ contiguous memory

drawbacks → contiguous memory
fixed space.

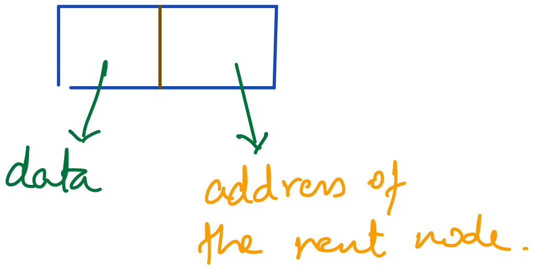


Since no contiguous memory available to create array, we have to device a way to use chunks of memory for same purpose.

To solve this, LL was created.

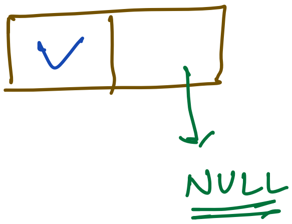
Node

NODE

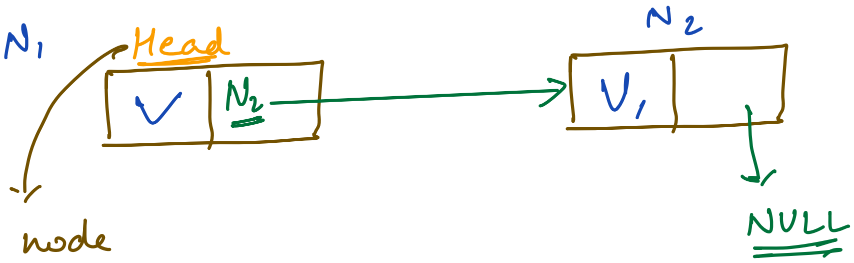


Structure of Node

```
class Node
{
    datatype value.
    Node next
    Node ( datatype v)
    {
        value = v ;
        next = NULL
    }
}
```



How to point N_1 to N_2.



first node of L.L e/a HeaD

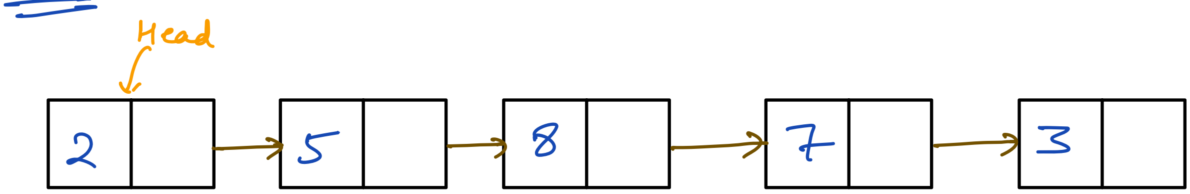
$N_1.next = N_2$

Operations on L.L

1) Access K^{th} element (K = 0 is first element)

Array → A[K] ⇒ O(1)

L.L



```
Node temp = head.
for (1 → K)
{
    temp = temp.next
}
return temp.val
```

* Never update head b/c you will lose the L.L

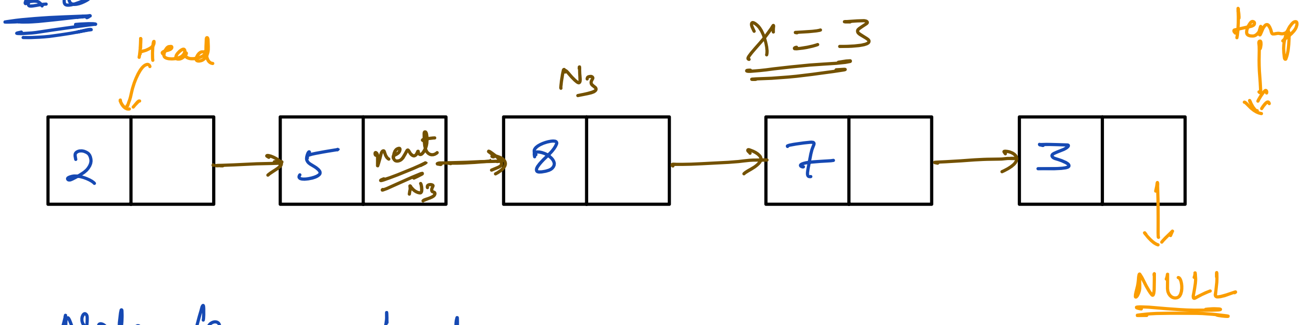
T.C ⇒ O(K)

2) Check for value X (searching)

Arrays → 1) Linear Search → O(N)

2) Binary Search → O(log(N)) → only sorted data.

L.L



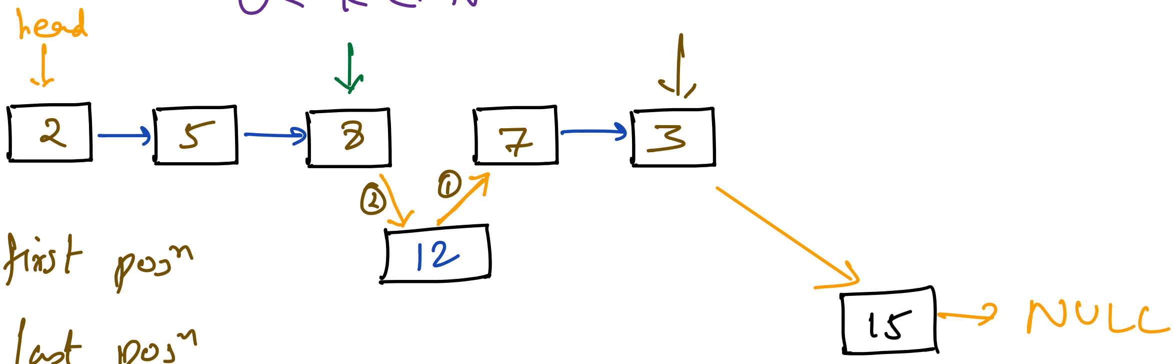
```
Node temp = head.
while ( temp != NULL)
{
    if( temp.val == X)
        return temp/true.
    temp = temp.next
}
return false
```

T.C ⇒ O(N)

3) Insertion

Insert node X at K^{th} posn (0-based)

$0 \leq K \leq N$



(c-1) first posn

(c-2) last posn

(c-3) anything in b/w

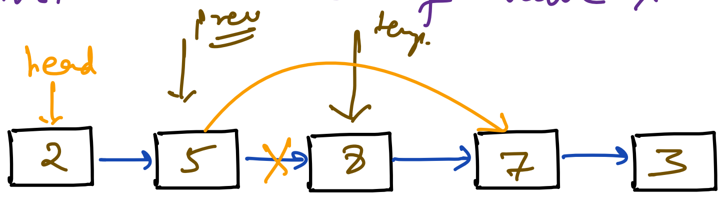
K=0

```
if ( K == 0 )
{
    newNode.next = head
    head = newNode *
}
else
{
    temp = head;
    for ( i -> 1 to K-1 )
    {
        temp = temp.next  ---> move to next node.
    }
    temp.next = newNode.
    newNode.next = temp.next  } wrong
    newNode.next = temp.next ✓
    temp.next = newNode ✓
}
```

T.C => O(K)

4) Deletion

delete first occurrene of value X in given L·L , delete it.



x=2

Edge cases

1) L·L is empty ⇒ Head → NULL ✓

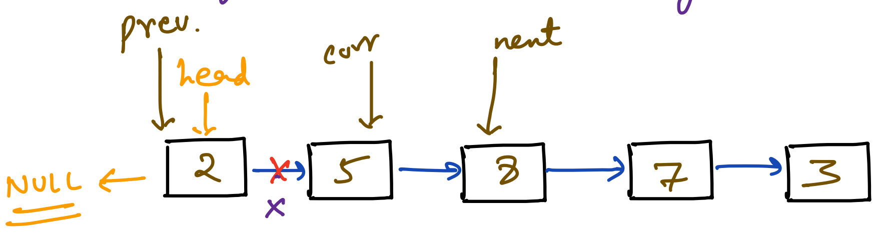
2) L·L only has 1/2 node.

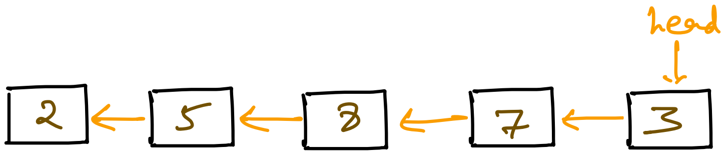
Code

```
if( Head == NULL)  return  Head.
if( head.val == X)
{
    Head  =  head.next
    return Head
}
Node temp = head
while ( temp.next!=NULL)
{
    if( temp.next.data == X)
    {
        temp.next  =  temp.next.next
        return Head
    }
    temp = temp.next
}
return Head.
```

T.C ⇒ O(N)

Q Reverse the given linked list by updating pointers S.C => O(1)





Head curr.next

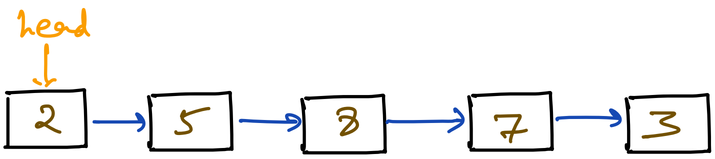
prev curr next

NULL

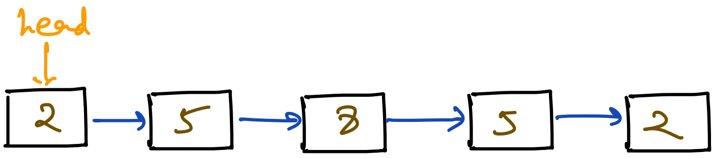
```
curr = Head.

prev = NULL
while ( curr != NULL)
{
    next = curr.next
    curr.next = prev.
    prev = curr.
    curr = next
}
head = prev
```

T.C => O(N)

Q Check if the given L.L is a palindrome.



Ans = false



Ans = True

Sol 1 Create a copy of linked list & reverse it.

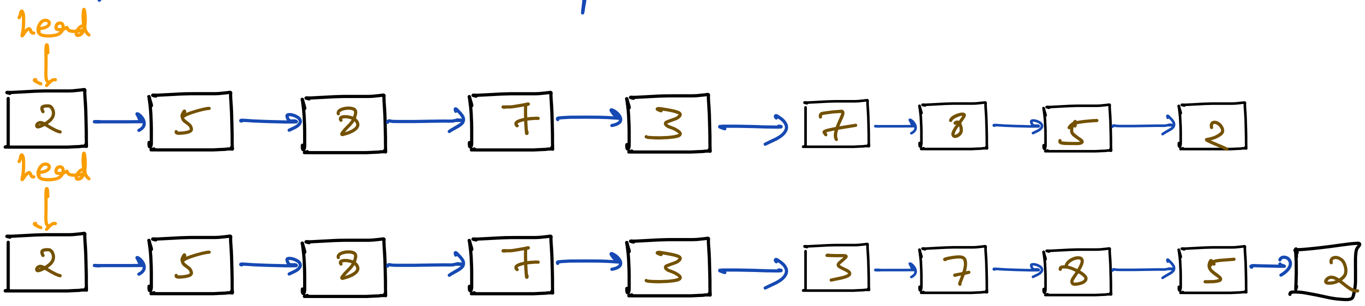
T.C → O(N + N + N) = O(N)

S.C ⇒ O(N)

Sol 2 1) Find mid → O(N)

2) Reverse 2nd half. → O(N)

3) Traverse & compare. → O(N)



```
1) // Mid element
   n=0
   temp = head
   while( temp != NULL)
   {
      n++
      temp = temp. next
   }
2) // Go to middle..
   temp= head.
   for ( i → 1 to n/2)
   {
      temp= temp.next
   }
3) Reverse
4) Compare. → careful with last node
```