LinkedList 1

Good Evening

* Agenda for Today

01. Basics of Linkedlist
02. Accessing of element
03. Searching for a value
04. Insertion in LL
05. Deletion in LL
06. Reverse a Linkedlist (vvI)
07. Check if palindrome

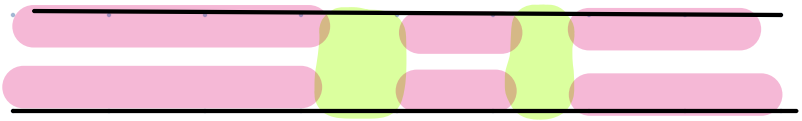
LinkedList → Linear data structure

Issues with Array

→ Fixed size

→ Costly insert & delete operation

→ Continous memory location



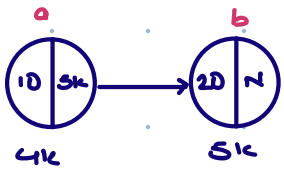
Linkedlist → Utilise all the free memory available wheter its continous or non continous

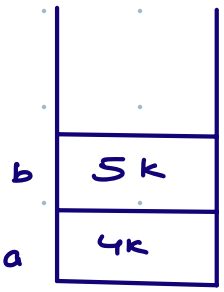
→ Insertion & Deletion in Linkedlist is optimal as compared to Arrays.

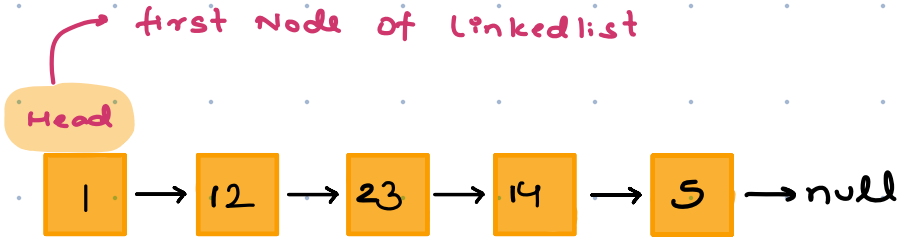

```
class Node {
    int data;
    Node next;

    Node (int d) {
        data = d;
        next = null;
    }
}
```

```
Node a = new Node(10);
Node b = new Node(20);
```

```
a.next = b
```







- Any linked list can be represent by the first Node or head.

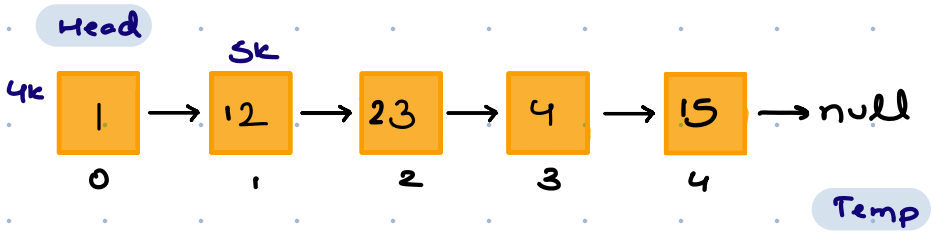
```
int size (Node head){
}
```

→ Question signature

* Never move head pointer
* Null pointer Exception

* Operation on Linkedlist

01. Access the K' element (K=0 → represents first element)



6

K = 3 ⟶ Ans = 4

K = 2 ⟶ Ans = 23

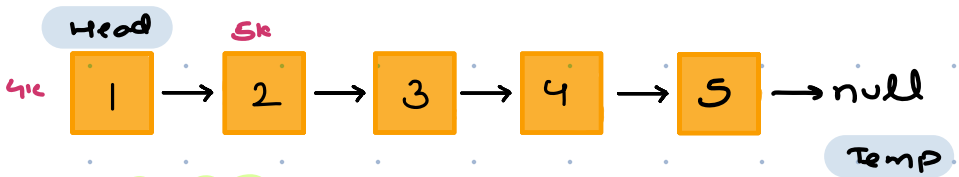
K = 6 ⟶ Ans = -1

```
if (head == null) return -1;

Node temp = head;

for (i=0; i<k; i++){
    if (temp == null) return -1;
    temp = temp.next
}

return temp.data;
```

TC : O(N)
SC : O(1)

* Check for Value (X) (Searching)

X
value



X = 3 ⟶ Ans = true

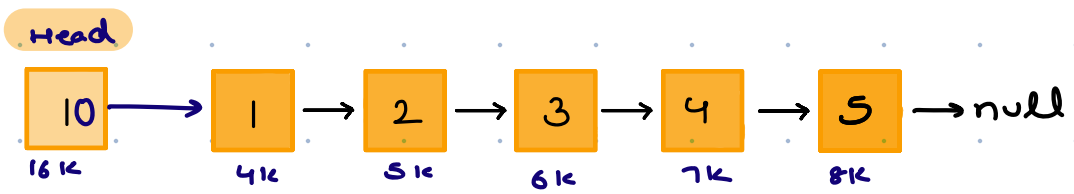
```
if (head == null) return false;
Node temp = head;
while (temp != null) {
    if (temp.data == x) return true;
    temp = temp.next;
}
return false;
```

TC : O(n)
SC : O(1)

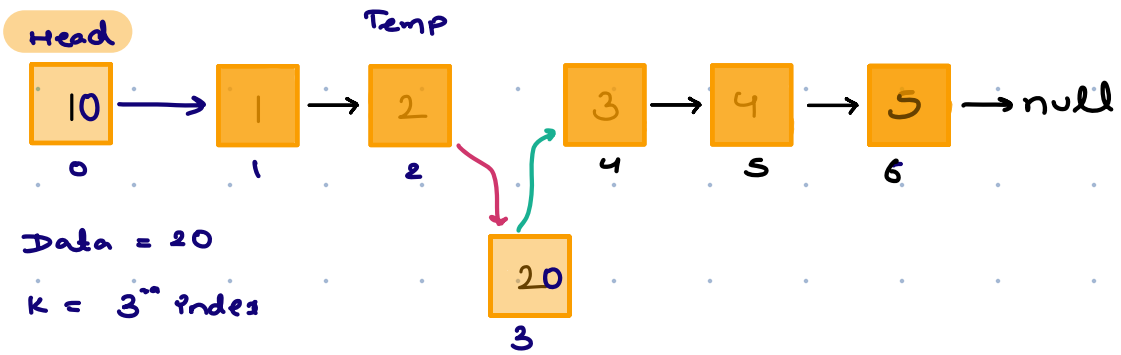
* Insert a new Node with Data at K^th position

Data = 10
K = 0

} Add it in front




```
Node nn = new Node(data);
nn.next = head;
head = nn;
```



```
Node nn = new Node(20);

Node temp = head
 i = 0
while ( i < k-1 ){
    temp = temp.next
    i++;
}

nn.next = temp.next

temp.next = nn;
```

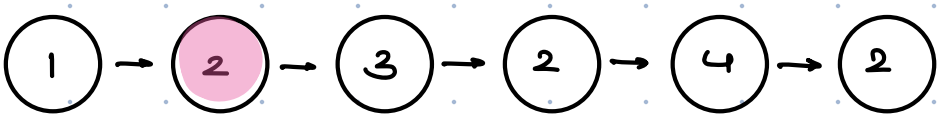
Edge cases

01. Null case while traversing

02 Head is null

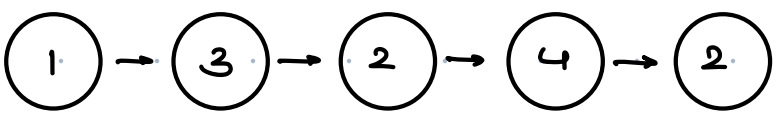
* Deletion in Linkedlist

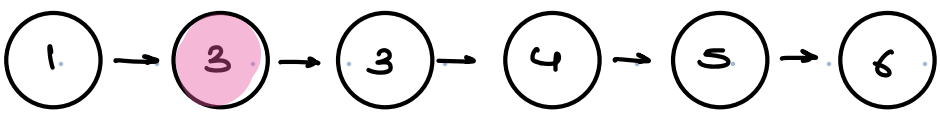
→ Delete the first occurrence of value x given in Linkedlist. If x is not present, leave LL as it is.



x = 2

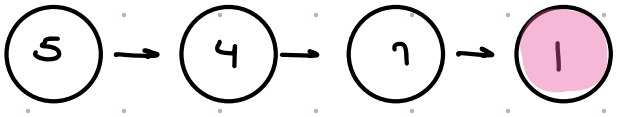
Ans ⇒





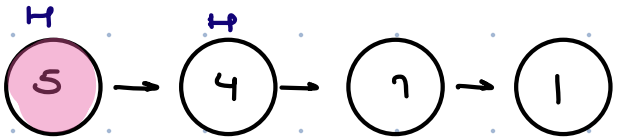
x = 3

Ans =



$x = 1$

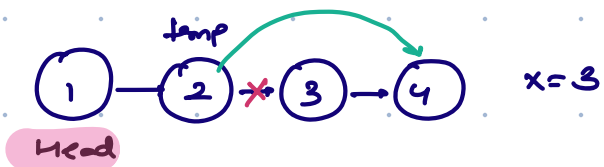
Ans =



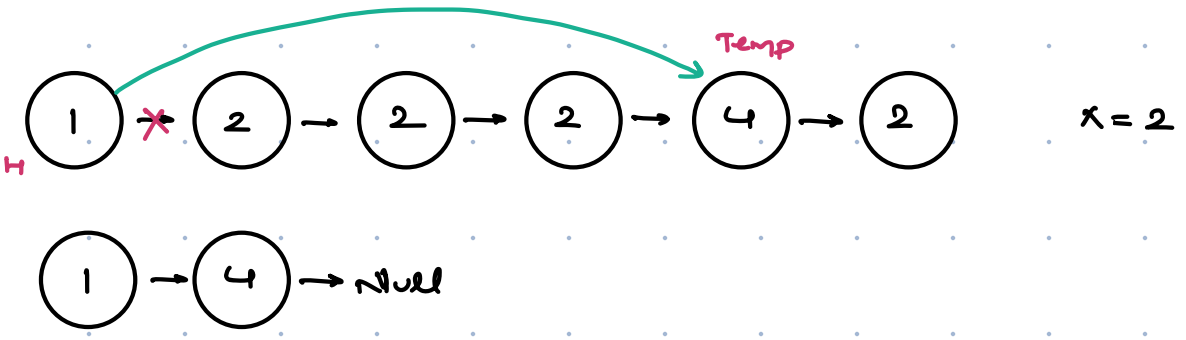
$x = 5$

Ans =


```
Node deleteFirst (Node head){
    If (head == null) return head;

    if (head.data == x){
        head = head.next
        return head;
    }
    temp = head
    while (temp != null){
        if (temp.next != null && temp.next.data == x)
            temp.next = temp.next.next;
            return head;
        }
        temp = temp.next;
    }
    return head;
}
```

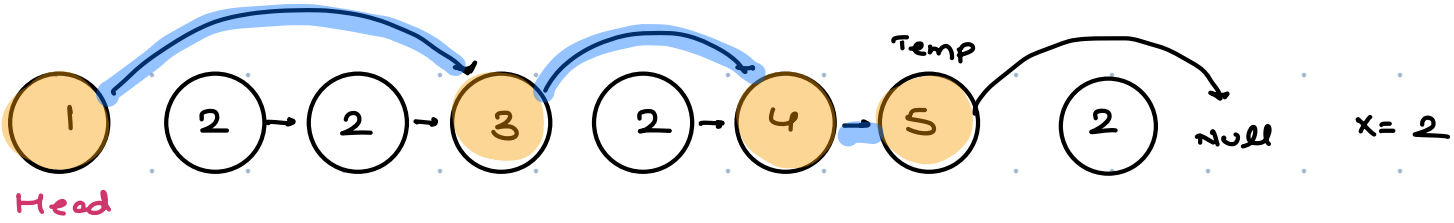


* Delete all occurrences of x

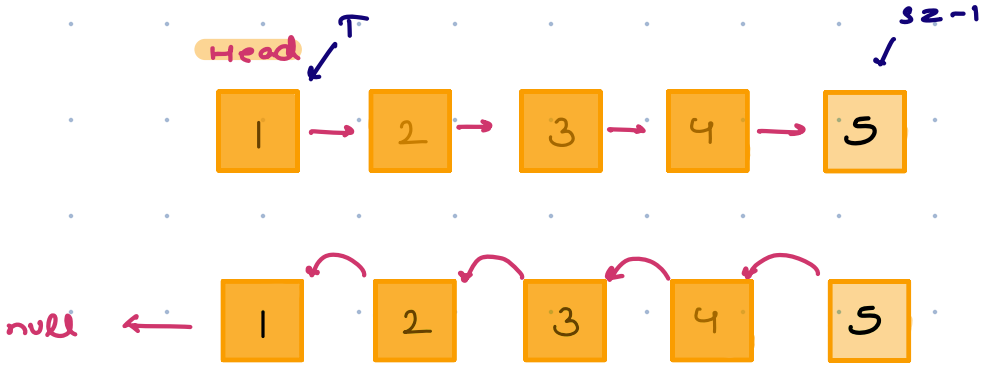


Idea → check for all value of x & move your next ptr accordingly

```
while (head != null && head.data == x){
    head = head.next;
}

Node temp = head;

while (temp != null && temp.next != null){

    if (temp.next.data == x){
        temp.next = temp.next.next;
    }
    else{
        temp = temp.next;
    }
}

return head;
```



* Reverse a Linkedlist



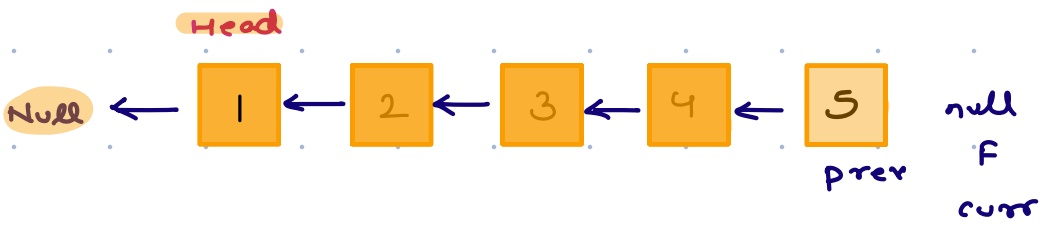
* Brute force

01. Size of LL

02. temp= 0
last = sz-1
} swapping

03 temp= 1
last = sz-2
} swapping

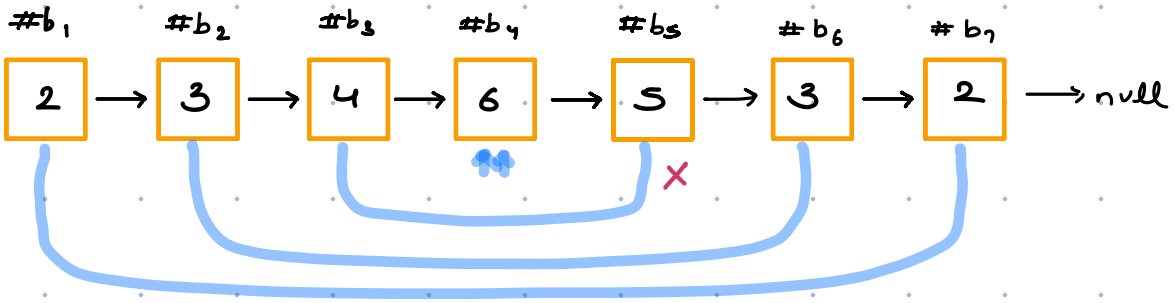
* Use 3 pointers



```
Node prev = null;
Node curr = head

while (curr != null) {
    Node forw = curr.next
    curr.next = prev
    prev = curr
    curr = forw
}
return prev;
```

Tc : O(N)
Sc : O(1)

* Given a Linkedlist, check if it is a palindrome.



Not a palindrome

{ ACA, Naman, Maam }

(1) = Valid palindrome

Solution 1

→ Reverse the entire linkedlist

→ Compare the old linkedlist & reversed LL.

Solution 2

→ Get the mid of Linkedlist ⟶ size/2

→ Reverse the second half

→ Compare the first half with reversed second half.

Try this Approach