Linked List 1

~~~~~~~
Agenda
~~~~~~~

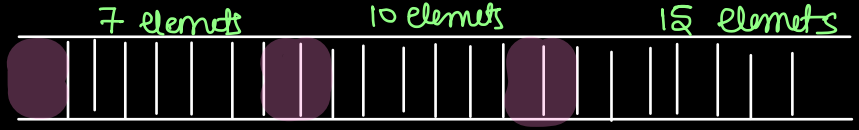
1. Introduction to LinkedList
2. Operations on LinkedList :
   * Access Kth Element
   * Searching
3. Insert New Node
4. Deletion in LinkedList
5. OnePlus removes Defects
6. Reverse the LinkedList
7. Check Palindrome

Hello Everyone
Very Special Good Evening
to all of you 😊😊😊
We will start session
from 9:06 PM

~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Introduction to LinkedList
~~~~~~~~~~~~~~~~~~~~~~~~~~~~

(+ continuous memory based data structure)

Issues with Array :



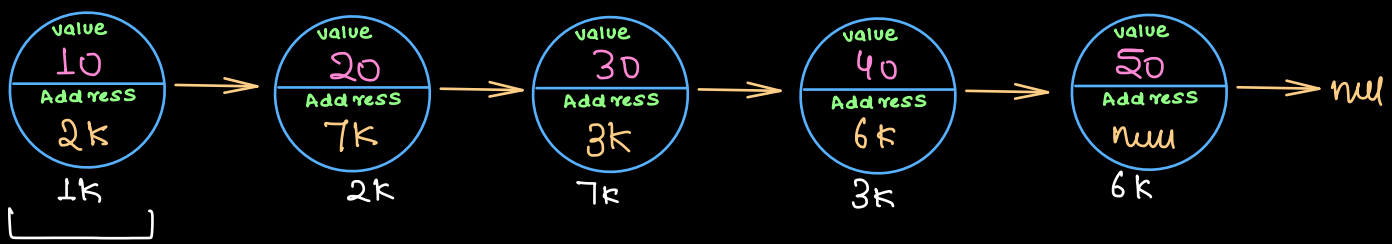
max size for array which we can create here is '15'

→ Issue: Array requires continuous memory allocation.

Linked List :

* LinkedList(LL) is linear data structure that can utilize all free memory.
* Need not to have continuous memory to store the data in LL.

Representation of LinkedList :




```
class Node {
    int data;      → data that we want to store
    Node next;
    // constructor
}
```

Student

Rollno	→ int
Name	→ String
marks	→ double
Details of Sibling	→ Student

Store data of multiple students:

User defined Datatype:

```
class Student {
    int roll_no;
    String name;
    double marks;
    Student siblings;
}
```

} blueprint of Student data type is ready.

ref var = 4k

```
Student S1 = new Student();
S1.rollno = 1;
S1.name = "ABC";
S1.marks = 95.6;
S1.sibling = S2;
```

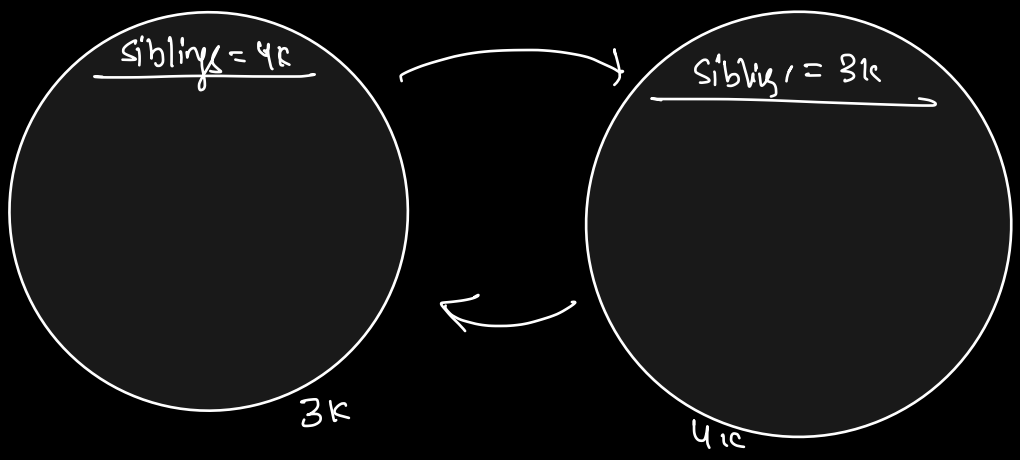
S1.sibling = S2; → 3k

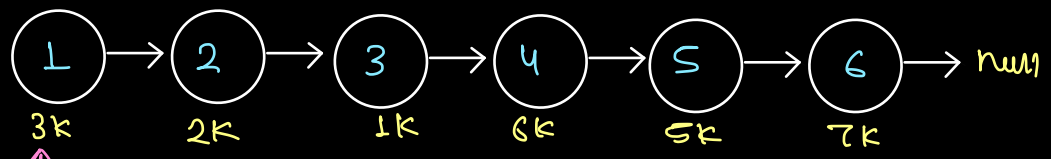
ref var = 31c

```
Student S2 = new Student();
S2.rollno = 2;
S2.name = "BCD";
S2.marks = 93.2;
S2.sibling = S1
```

S2.sibling = S1 → 4k

S1 is sibling of S2 & vice versa





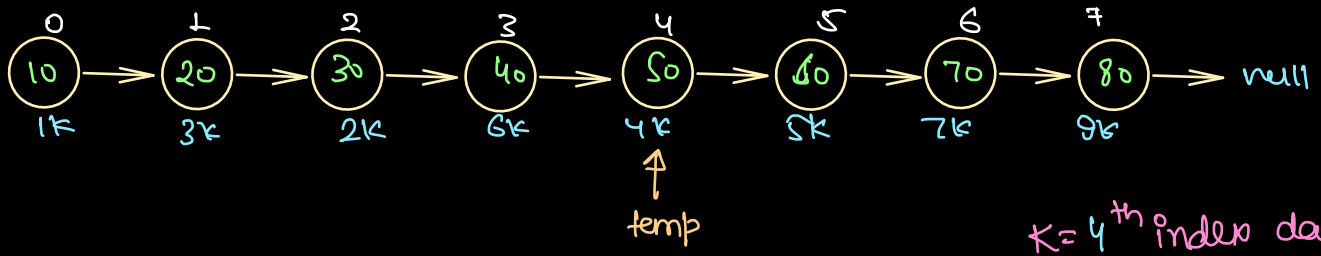
* first node of LL is called Head
* from Head only, we can represent entire LinkedList.

~~~~~~~~~~~~~~~~~~~~~~~~

Operations on LinkedList :

~~~~~~~~~~~~~~~~~~~~~~~~

* Access kth element(k = 0; k is the first element)



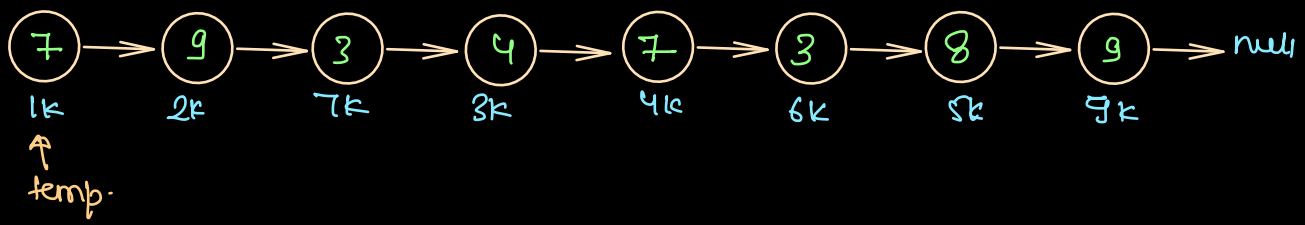
K = 4th index data

Head = 1K

```
Node temp = head;
for(int i=0; i<K; i++) {
    temp = temp.next;
}
```

i	i<K	temp=temp	i++
0	T	3K	1
1	T	2K	2
2	T	6K	3
3	T	4K	4
4	F	loop stops.	

* Check for value X (searching)



Head = 1K

X = 3 - present in it or not

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

T.C: O(n)

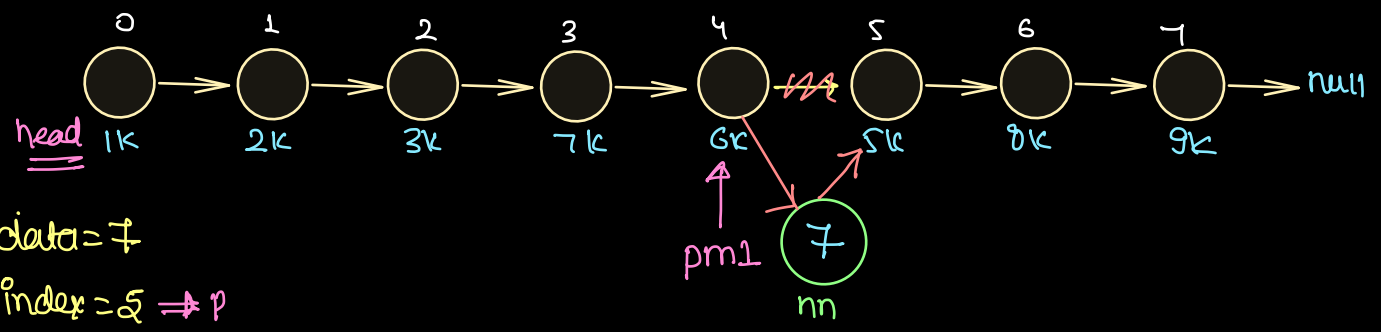
S.C: O(1)

Insert New Node

Implementation:
→ Doubt Section
(in JAVA)

Insert a new node with data v at index p in the linked list

Though indexing doesn't exist is LL, but for our understanding, let's say Node 1 is at index 0, Node 2 at index 1, etc.



data = 7
index = 5 ⇒ p

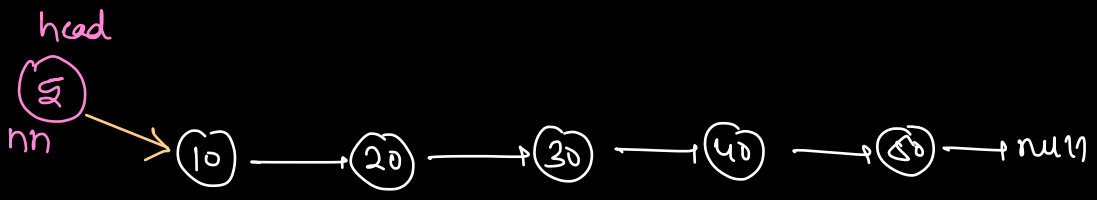
Steps:

① find $(p-1)^{th}$ node Node pm1 = findkthNode(head, p-1);

② Create new Node Node nn = new Node(7);

③ Make connections properly;

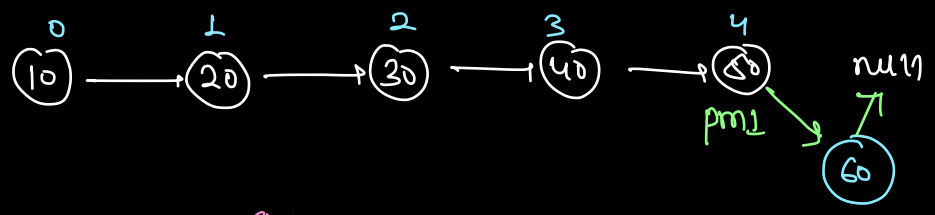
```
nn.next = pm1.next;
pm1.next = nn;
```



data = 5
p) index = 0
} Edge case → Handle it seperately.

```
Node nn = new Node(data);
if( p == 0) {
    nn.next = head;
    head = nn;
}
```

Insertion at last index;



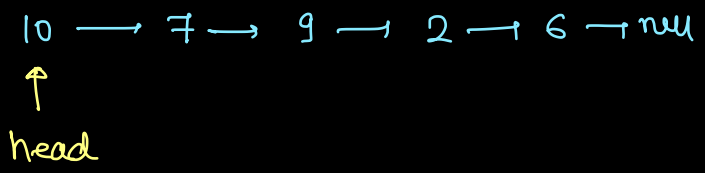
data = 60
Indx(p) = 5

Steps:

1. find $(p-1)^{th}$ node — Node pm1 = findkthNode(head, p-1);
2. Create new Node — Node nn = new Node(7);
3. Make connections properly;

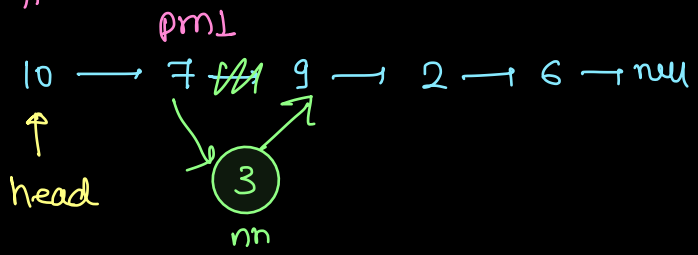
```
nn.next = pm1.next;
pm1.next = nn;
```

Case → first node
middle part
last node



data = 3
index = 2

o/p:

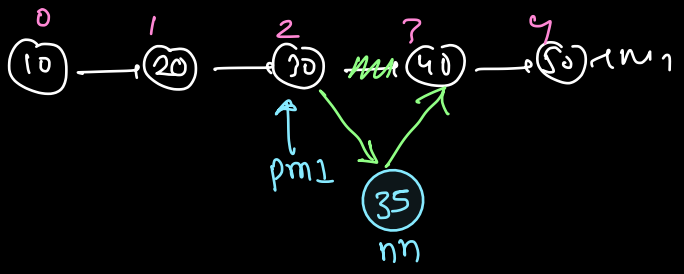


1. find 1st index node → pm1 = $(p-1)^{th}$ node.
2. Create a new node → Node nn = new Node(3)
3. Connection →

```
nn.next = pm1.next;
pm1.next = nn;
```

pseudocode:

```
Function InsertAtIndex( head , data, p) {
    Node nn = new Node(data);
    if( p==0) {
        nn.next=head;
        head=nn;
    }
    else {
        //find (p-1)th node using function: Above discuss probe.
        Node pm1 = findKthNode( head, p-1);
        nn.next= pm1.next;
        pm1.next =nn;
    }
    return head;
}
```



data = 35
index = 3

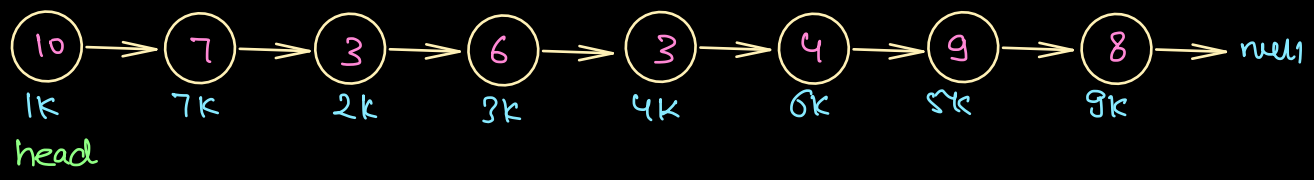
10:44 - 10:54 → Break.

* OOPs: classes & object
  → Revise
  → Revise memory

follow: Good listener: Attention.
Problem → Doubts.

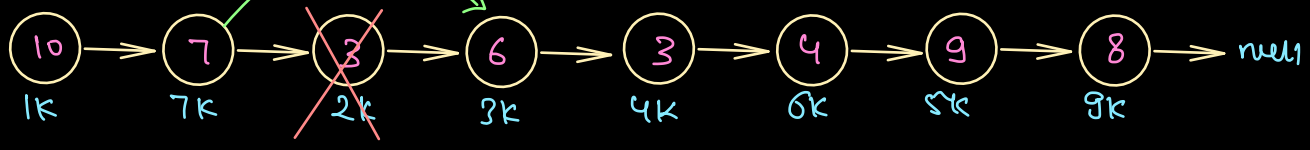
Deletion in LinkedList

Delete the first occurrence of value X in the given linked list. If element is not present, leave as is.



X=3

O/P:

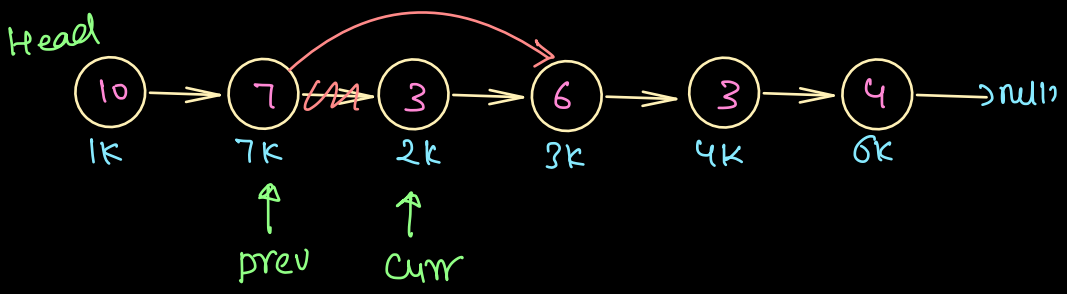


Approach:

Assumption: We have previous node avl.

prev: previous of target node.

K=3



prev.next = curr.next

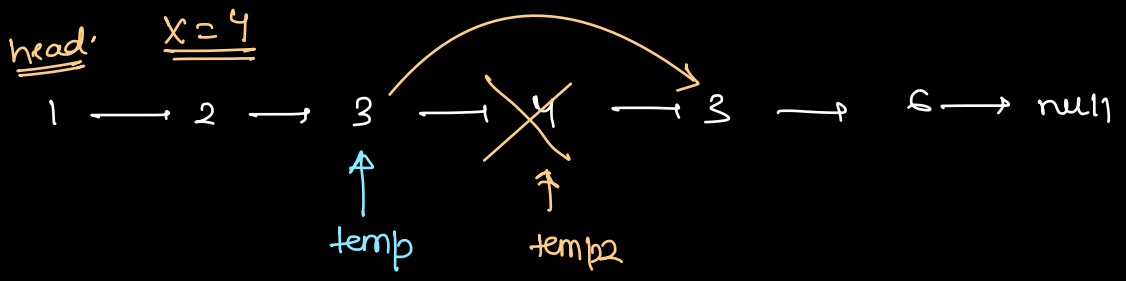
7K.next = 3K

X=1

5 → 4 → 7 → X → null

How to previous node?

O/P: 5 → 4 → 7 → null



temp.next.data → target ??

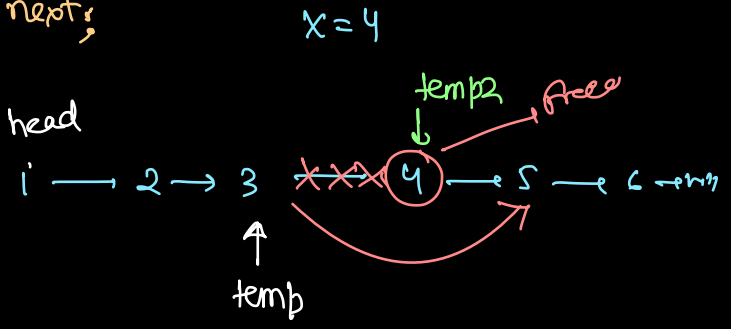
→ if true.

temp.next = temp.next.next;

(4).next

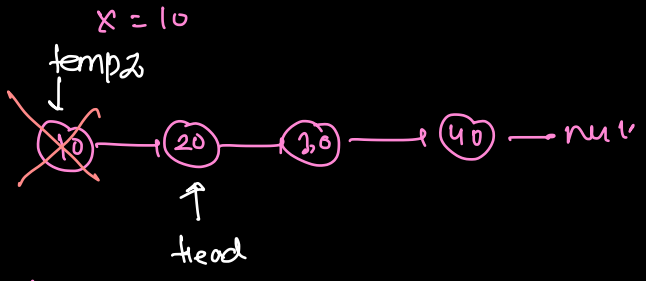
(3)

```
temp = head;
while(temp.next != null){
    if(temp.next.data == x){
        temp2 = temp.next;
        temp.next = temp.next.next;
        free(temp2);
        return head;
    }
    temp = temp.next;
}
```



Edge case: head.data == x → handle it separately.

```
if(head.data == x){
    temp2 = head;
    head = head.next;
    free(temp2); → in Java, not required.
    return head;
}
```



T.C: O(n)

S.C: O(1)

OnePlus removes Defects

Scenerio

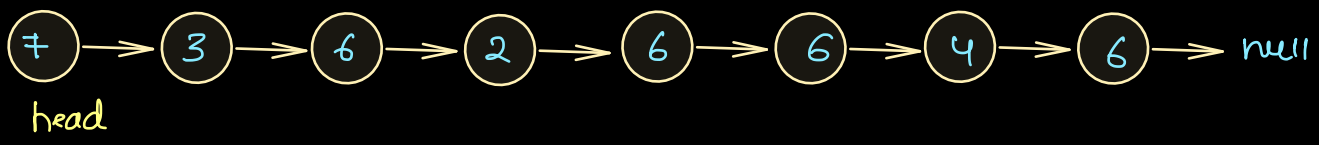
OnePlus has a lineup of N mobile phones ready in their manufacturing line. It has detected a defect in one of their phone models during production.
They have decided to recall all phones of the defective model from their manufacturing line.
Your task is to help OnePlus remove all defective phones from their production lineup efficiently.

Problem

You are given a linked list A of N nodes where each node represents a specific model type of a OnePlus mobile phone in the manufacturing line. Each node contains an integer representing the model number of the phone. You will also be given an integer B which represents the model number of the defective phone that needs to be removed.

Your goal is to remove all nodes (phones) from the linked list that have the model number B and return the modified linked list representing the updated manufacturing line.

B=6



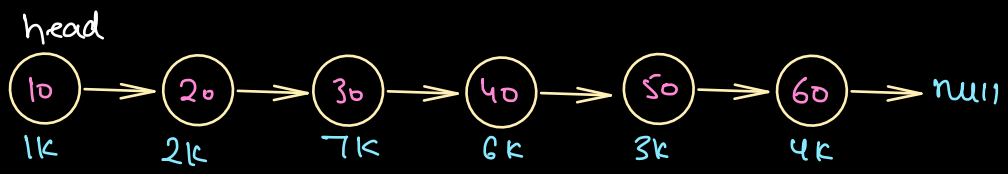
Delete all occurrence of B from the given linkedlist.

o/p:

TODO:

Reverse the LinkedList

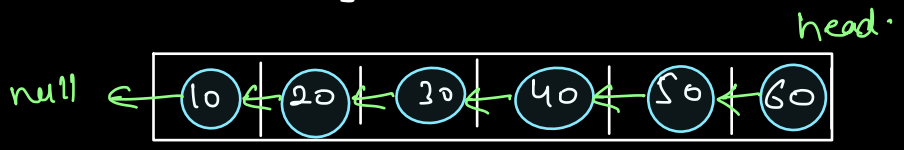
Given Head of a LinkedList, reverse the entire linkedlist.
Note: We can't use extra space. Manipulate the pointers only.



Idea 1

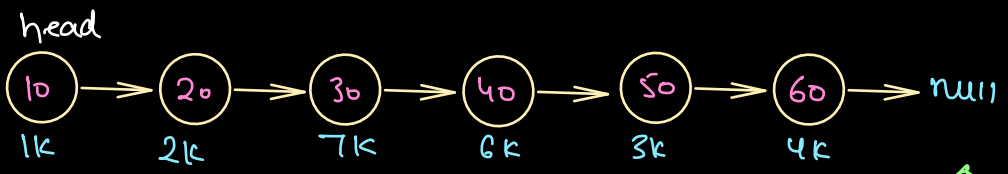
if extra space is allowed'.

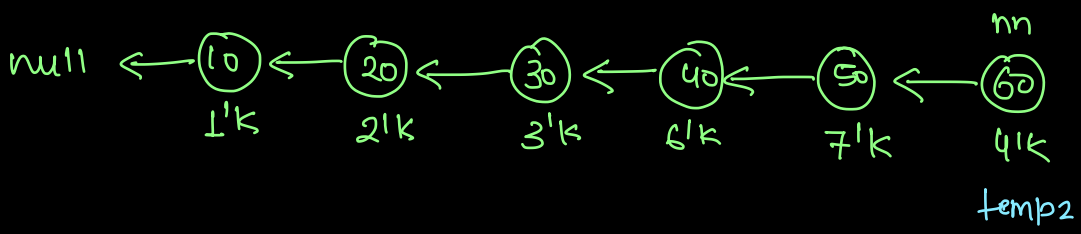
→ create array of nodes.



is it possible?
→ No, it is not issue in the array.

Idea 2:



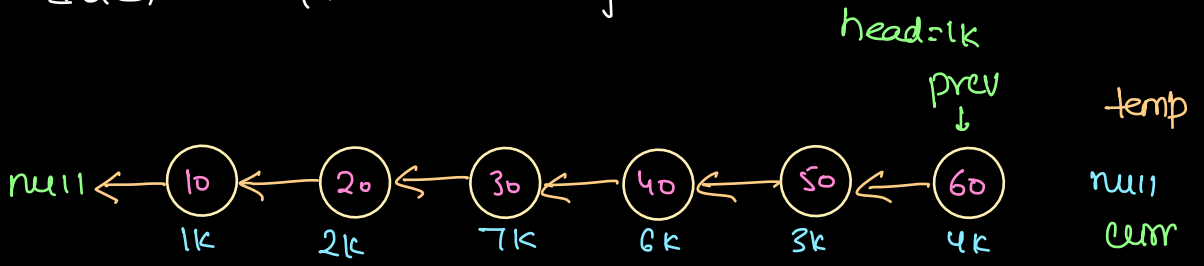



```
while( temp != null)
Node nn = new Node( temp.data);
nn.next = temp2;
temp2 = nn;
temp = temp.next
}
return temp2;
```

issue: taking Extra space & making changes in the new LL.

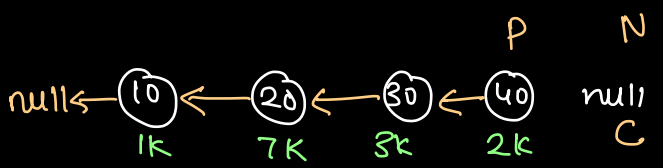
→ Do it in-place.

Idea 3: optimise way:

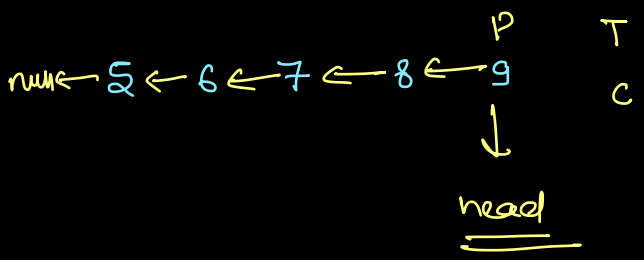



```
function reverse (head){
    prev = null;
    curr = head;
    while( curr != null) {
        temp = curr.next
        curr.next = prev;
        prev = curr;
        curr = temp;
    }
    return prev;
}
```

TC : O(n)



prev = ~~null~~ ~~1K~~ ~~7K~~ ~~3K~~ 2K
curr = ~~1K~~ ~~7K~~ ~~3K~~ ~~2K~~ null
next = ~~7K~~ ~~3K~~ ~~2K~~ null




```
prev = null;
curr = head;
while( curr != null) {
    temp = curr.next
    curr.next = prev;
    prev = curr;
    curr = temp;
}
return prev;
```

Check Palindrome

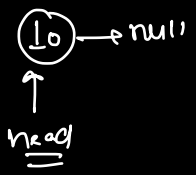
Given a Linked List, check if it is a palindrome.

malayalam → palindromic
madam → palindromic
naman → ?

o/p: true

o/p: true

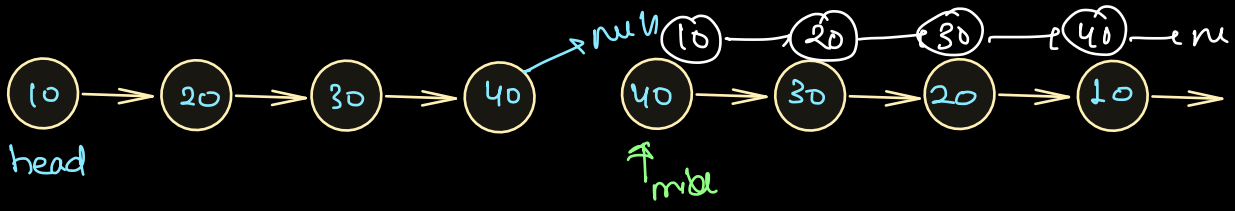
Single element → palindromic



Ideal: * Create a copy & reverse it
* Iterate and Compare

T.C: O(n)
S.C: O(n)

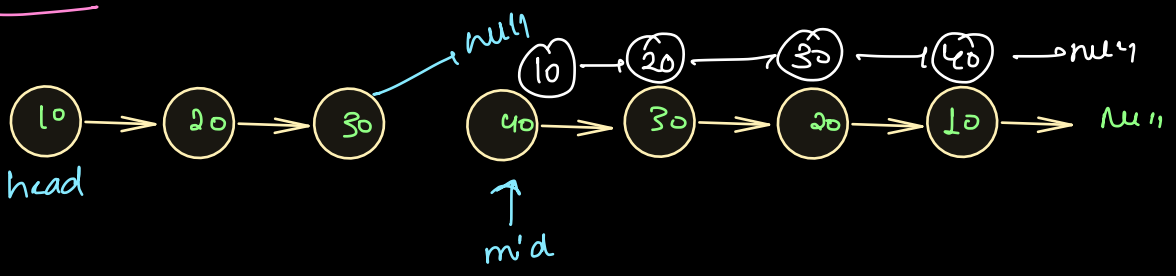
Idea2: optimise approach: (Even nods)



o/p: true

- Find mid of linkedlist
- Seperate it in two parts. (mid-1).next = null
- Reverse second part
- Compare palindromic or not.

odd:



o/p: true

it will work in odd length as well,

Implementation: TODO: