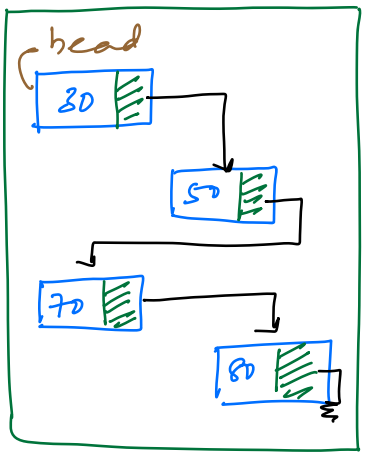
Today's Content

- → Linked List Basics
- → Size()
- → Get k^{Th} pos element
- → Insert(Node head, int pos, int val)
- → InsertSorted(Node head, int val)
- → Delete(Node head, int pos)
- → DeleteAllOcc(_____)

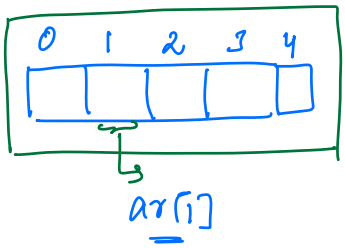
TODO
Try it out on your own.

- → Reverse k Nodes
- → Reverse k groups

Linked List Basics



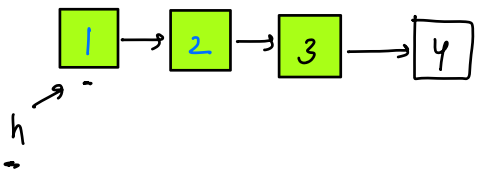
int ar[5]



TODO: In your language of choice learn the syntax.

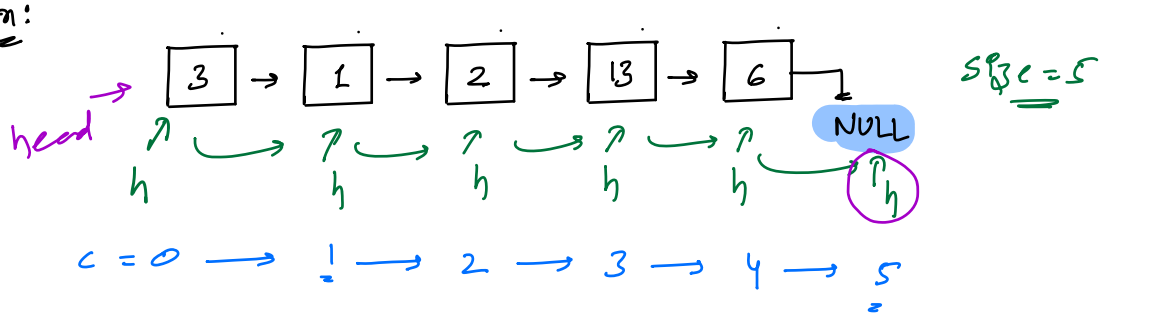
```
class Node {
    int data;
    Node next;
    Node(int n) {
        data = n
        next = NULL
    }
}
```

Ex: N = 4




```
Node h = new Node(1)
Node t = h
for(int i = 2; i <= N; i++) {
    t.next = new Node(i)
    t = t.next
}
return h
```

Eg:



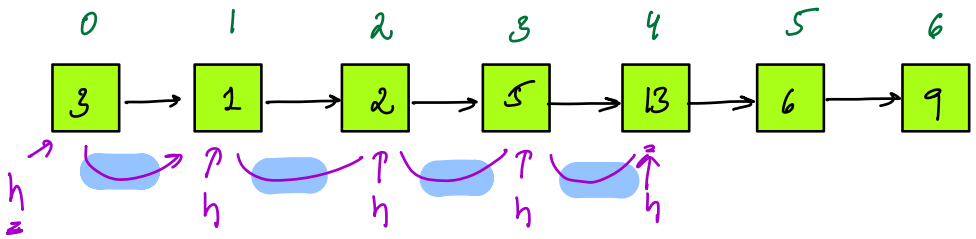

```
int size (Node h) {
    int c = 0; Node t = h
    while ( t != NULL) {
        c = c+1
        t = t.next
    }
    return c;
}
```

Eg: NULL : return 0

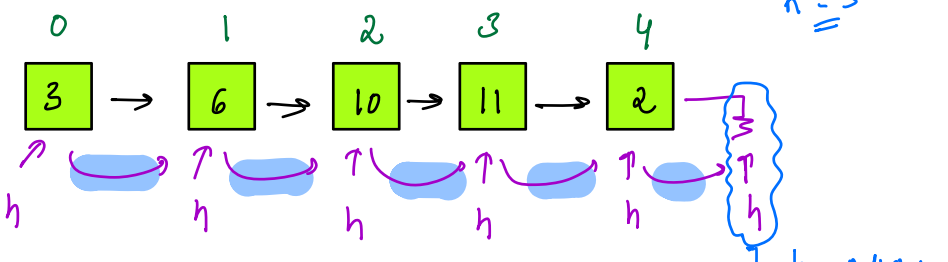
h

```
main ( ) {
    Node head = (We are creating Linked List)
        size (head)
}
```

Ex:



Ex:



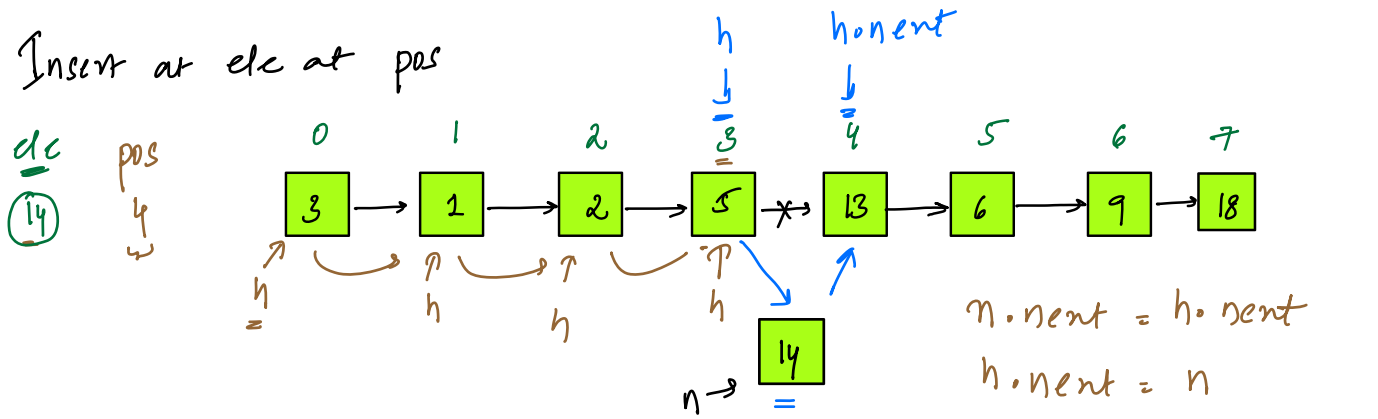

```
getKthElem(Node head, int k){          // k is 0 based index

    Node t = head;
    // update temp k times

    while( k > 0 && t != NULL ) {
        t = t.next
        k--;
    }

    if( t != NULL) { return t.data }
    else { // depends on Question, }
}
```

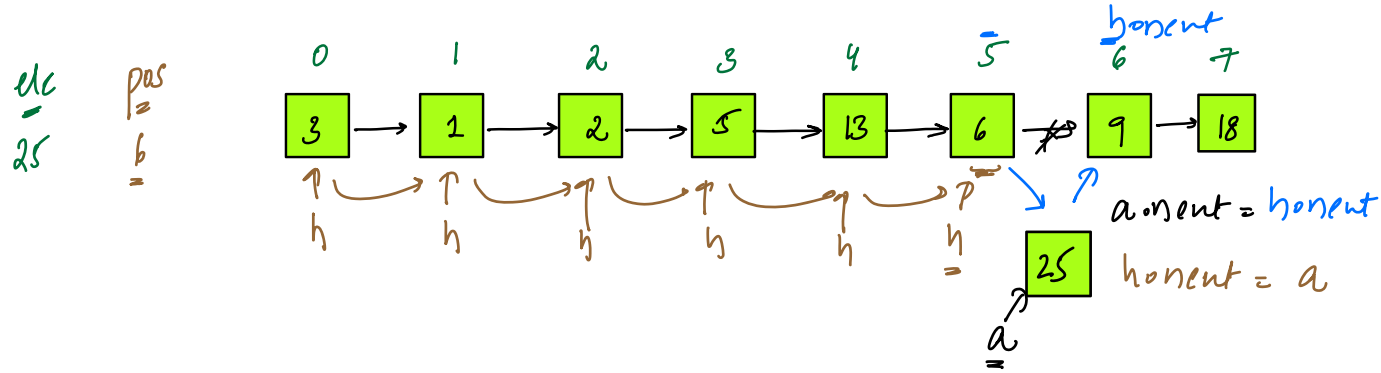
Edge case?

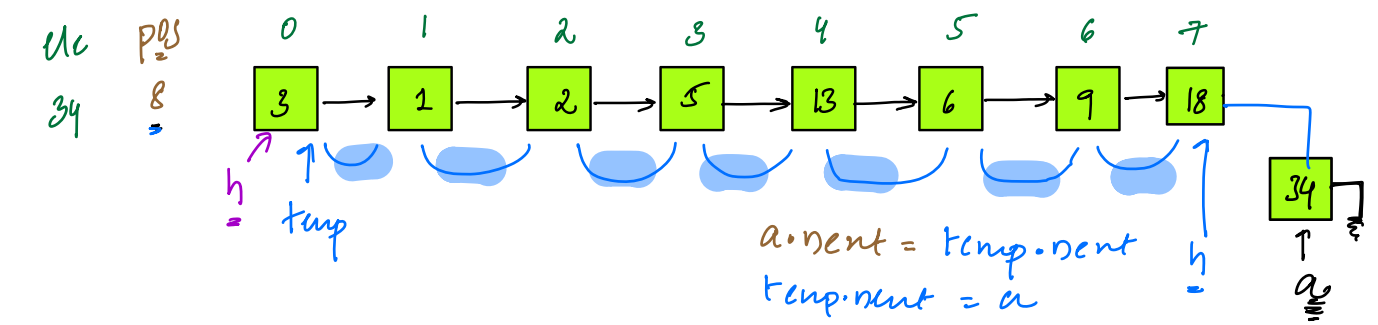
if k = 0 → t.data ✓

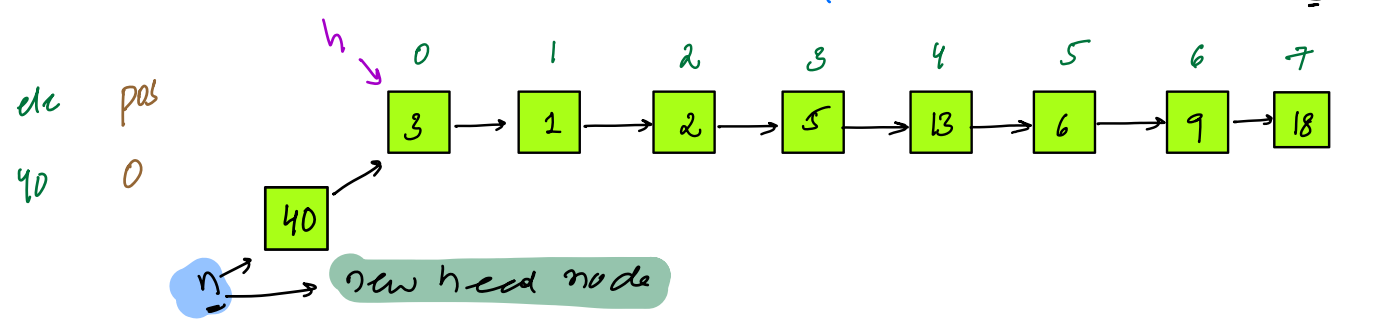
head == NULL
1) t = NULL
2) Break loop
3) Won't return t.data

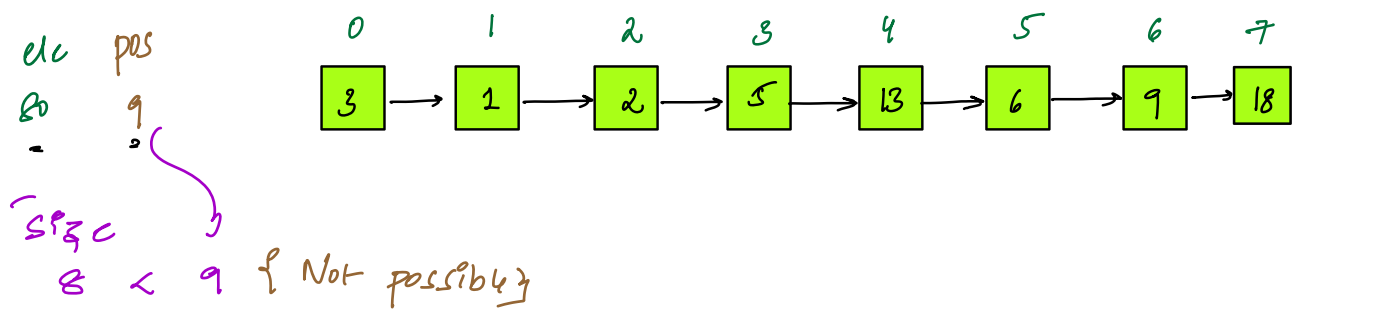
Insert at elc at pos



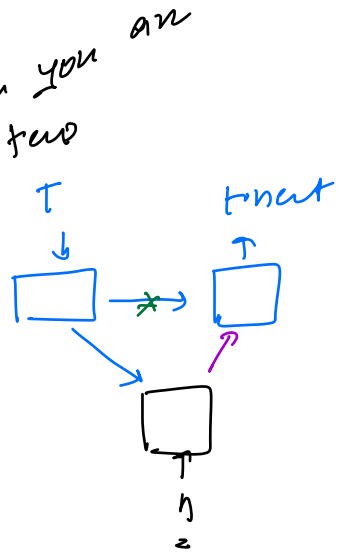




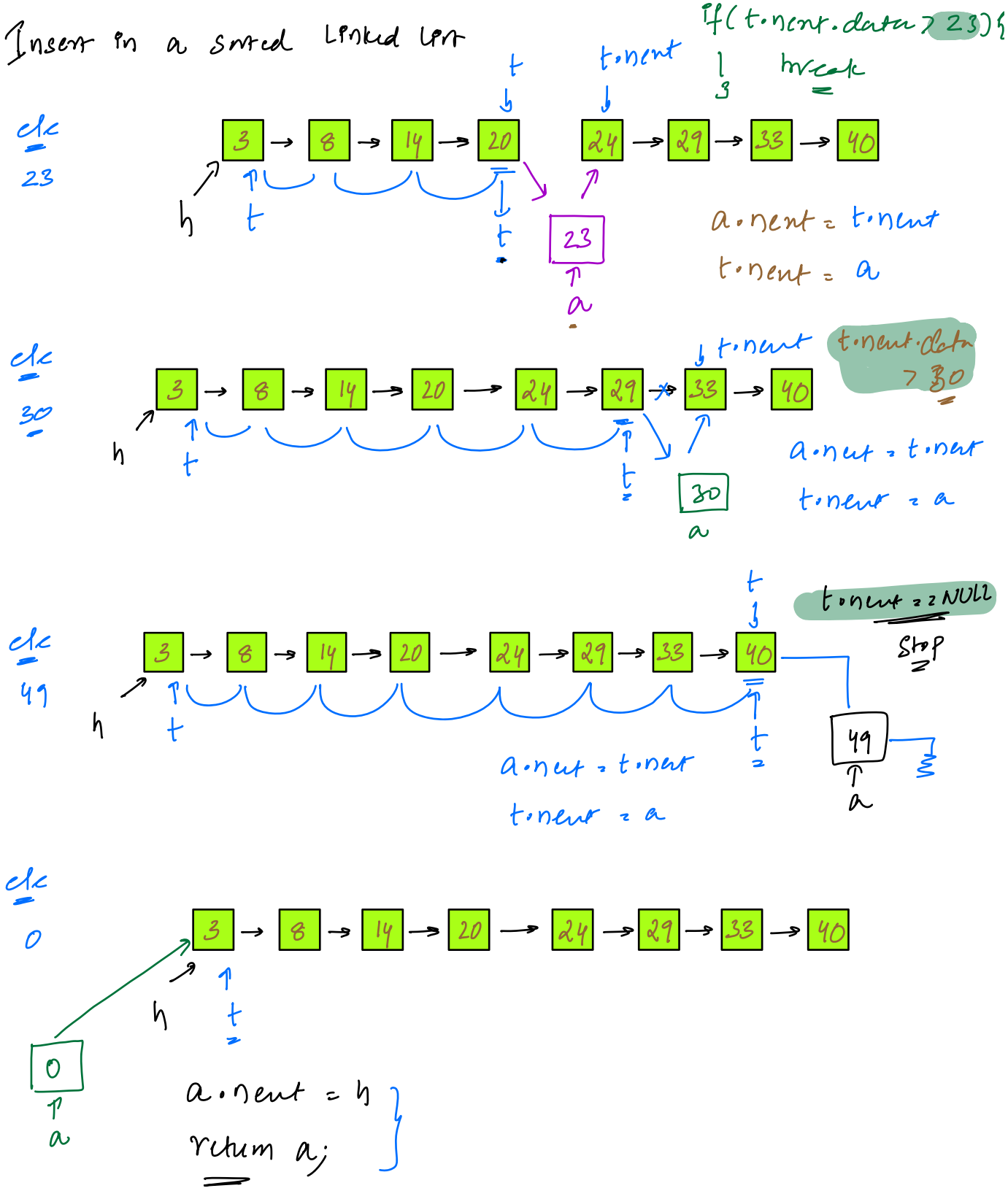




```
Node insert(Node head, int k, int ele){        // function should return head Node

    // Case-I { Not possible }
    if (k > size(head) || k < 0) { return head }

    Node n = new Node(ele) }                   // new node

    // Case-II { Insert at head }
    if (k == 0) {
        n.next = head
        return n;
    }

    // Case-III => { Can insert at kth pos
                     Insert at last }
    Node t = head;                             // how many times you are updating two
    for (int i=1; i<k; i++) {
        t = t.next
    }

    n.next = t.next
    t.next = n
    return t; }                                // not a head node

    return head;
}
```



Insert in a sorted Linked List



```
__ InsertSorted(Node head, int ele) {

    Node a = new Node(ele);
    Node t = head;

    // Case -1 // Insert at start

    if(t == NULL || ele < t.data) {
        a.next = h
        return a
    }

    while(t.next != NULL && t.next.data < ele){
        t = t.next
    }

    a.next = t.next
    t.next = a
    return head;
}
```

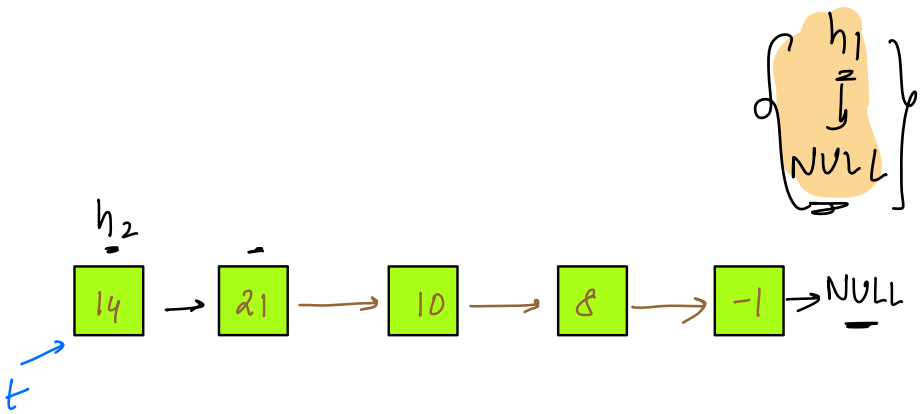
Edge Case
h = NULL
t = NULL
t.data
Error

We need to make sure t.next != NULL

Reverse Entire Linked List

Note: No Extra Space → SC : O(1)

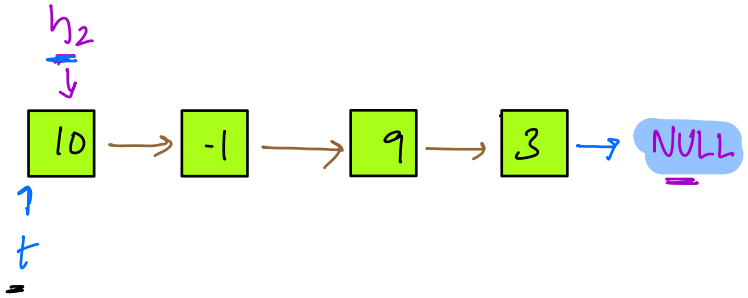
Note: We cannot change data




```
Node reverse(Node head) {
    h2 = NULL, h1 = head
    while( h1 != NULL) {
        t = h1
        h1 = h1.next
        t.next = NULL   { we can skip this line }
        t.next = h2
        h2 = t
    }
    return h2
}
```

TC : O(N)

Ex:



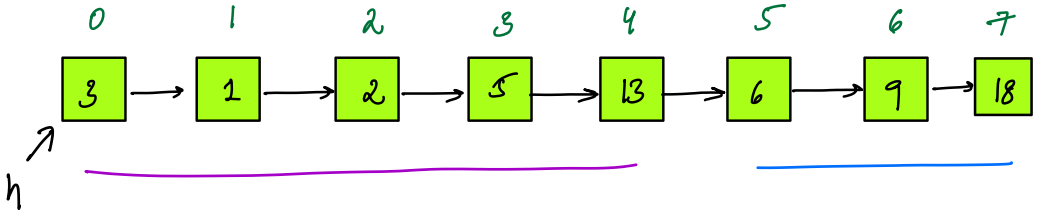
NULL
t
h_1

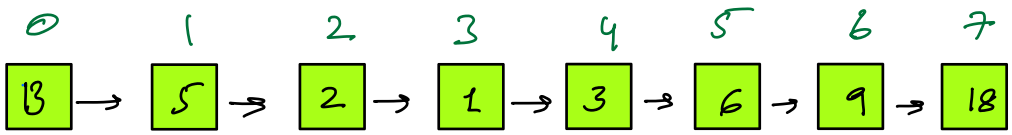
```
while( h1 != NULL){
    t = h1
    h1 = h1.next
    t.next = NULL
    t.next = h2
    h2 = t
}
```

Reverse first k Nodes of Linked List

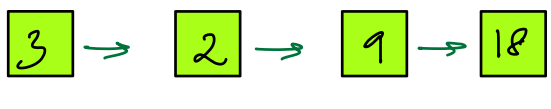
k → no: of nodes

k = 5



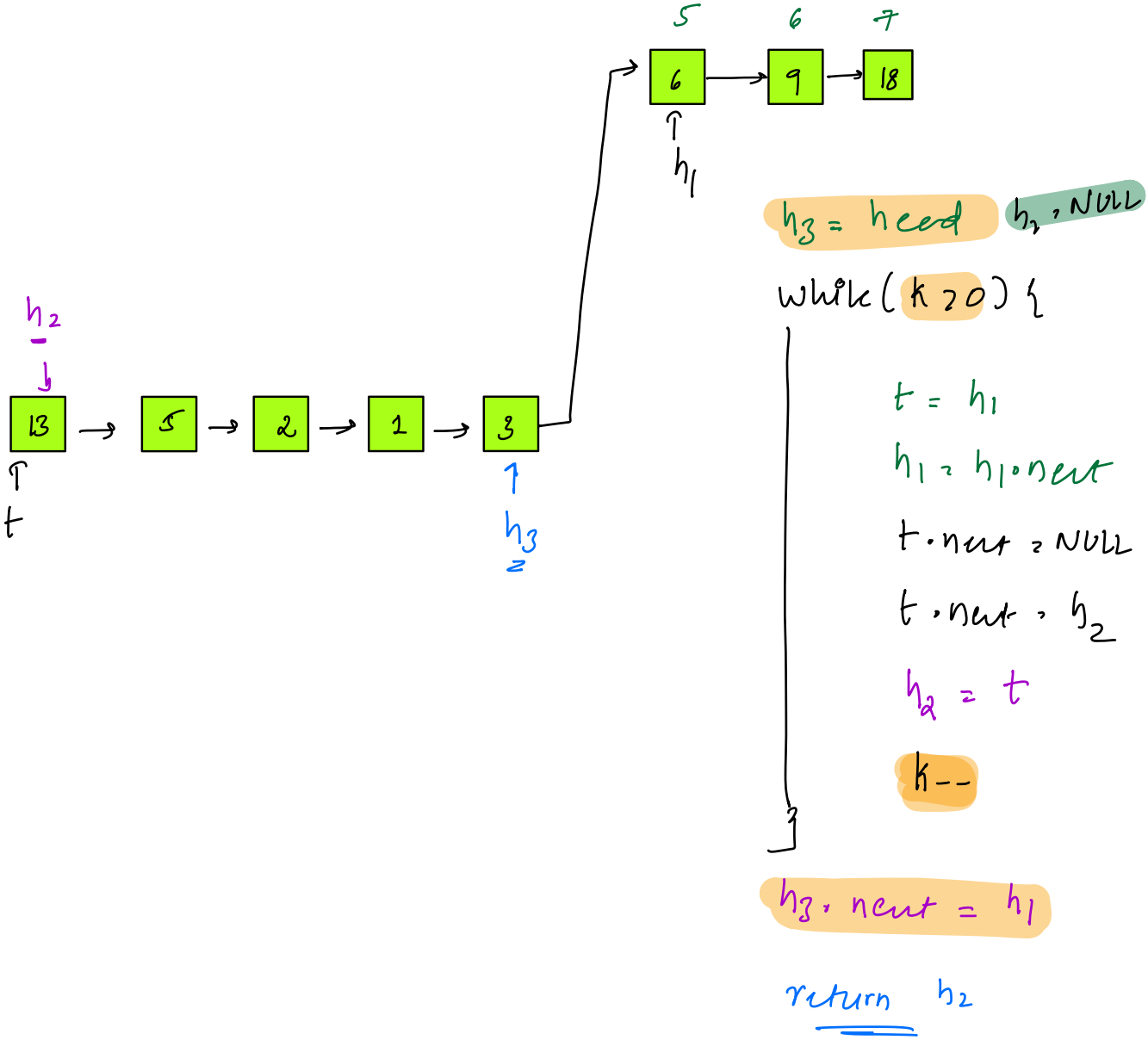


k = 5



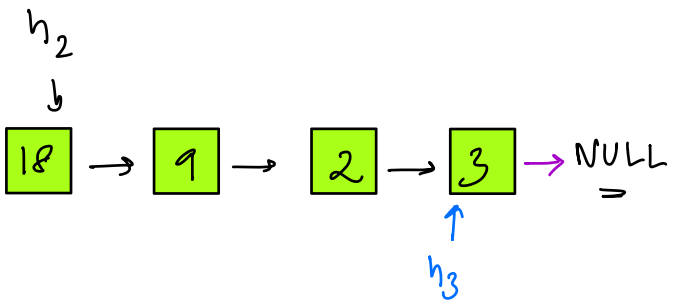
Expected output

18 → 9 → 2 → 3


5
6
7
6
9
18
h_1
$h_3 = head$
$h_2 = NULL$
h_2
13
5
2
1
3
t
h_3
while (k > 0) {
t = h_1
$h_1 = h_1.next$
t.next = NULL
t.next = h_2
$h_2 = t$
k--
}
$h_3.next = h_1$
return h_2

k = 5

NULL
↑ ↑
h_1 t



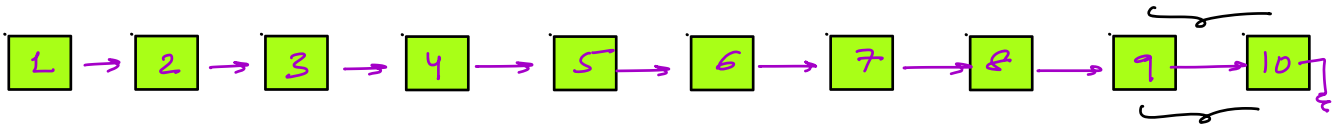

```
reverseFirstKNode(Node head, k) {
    if(k==0 || head==NULL) { return head }      // Edge Case - II
    h1 = head   h3 = head   h2 = NULL
    while(k>0 && h1!=NULL)                      // Edge Case I
    {
        t = h1
        h1 = h1.next
        t.next = NULL
        t.next = h2
        h2 = t
        k--
    }
    h3.next = h1
    return h2
}
```

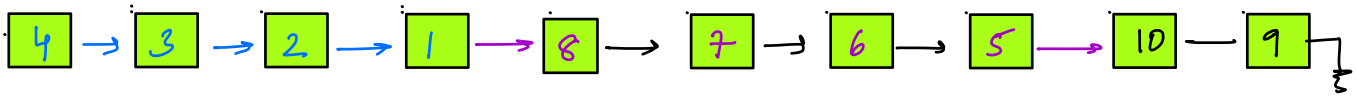
If k==0, don't even change list

// Google :

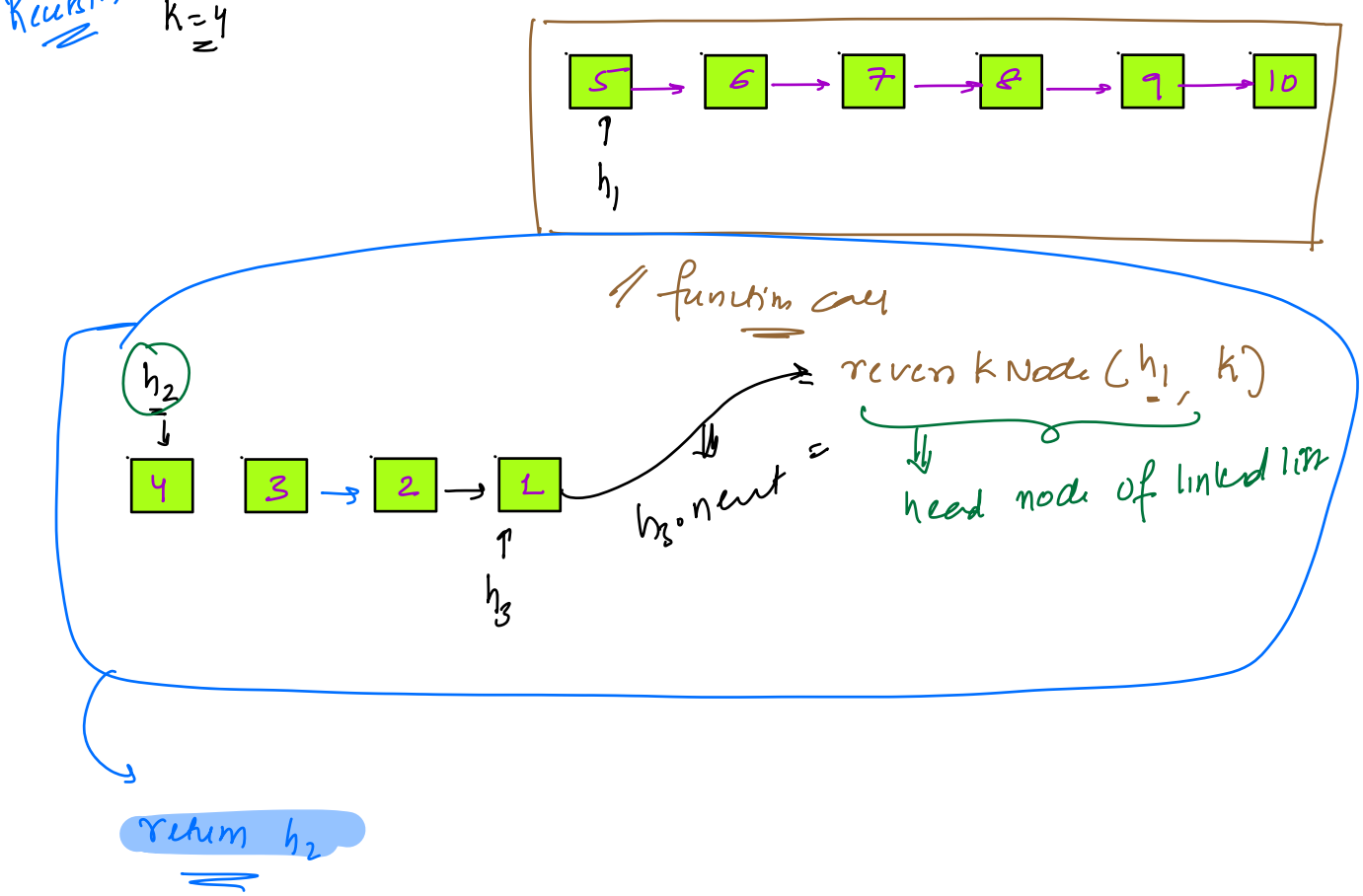
Reverse every k nodes of LinkedList

Ex: k = 4 → We cannot change nodes





Recursion k = 4



// Recursion:

Ass: Reverse k chunks of Nodes & return new Node

Main: ✓

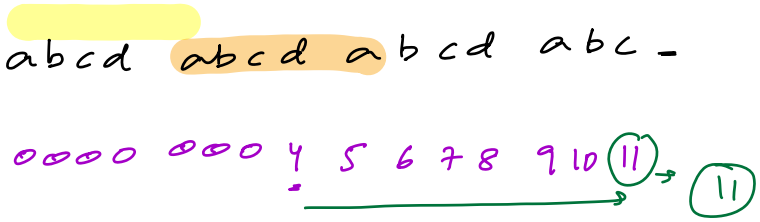
Base: =

```
Node ReverseKChunks(Node head, int k){
    if(head==NULL || k==0){return head;}
    h1 = head, h3 = head, h2 = NULL, tk = k

    // Reverse k Nodes
    while(k>0 && h1!=NULL)        // Edge Case 1
    {
        t = h1
        h1 = h1.next
        t.next = NULL
        t.next = h2
        h2 = t
        k--
    }
    h3.next = ReverseKChunks(h1, tk)
    return h2
}
```

Because, k is getting updated

doubts ?

Ex:



len = 4

abcda abcda abcda