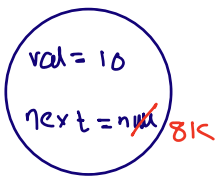
what is LinkedList : collection of Nodes

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

    Node ( int a ) {
        val = a;
    }
}
```

```
Node n1 = new Node (10);
Node n2 = new Node (15);
```



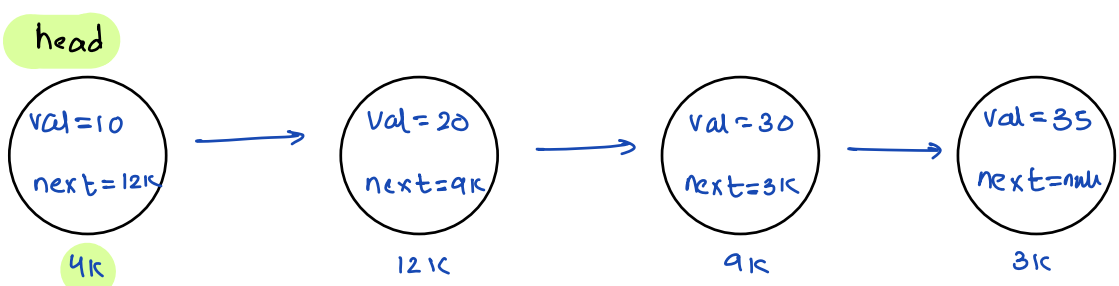
4K
(n1)

8K
(n2)

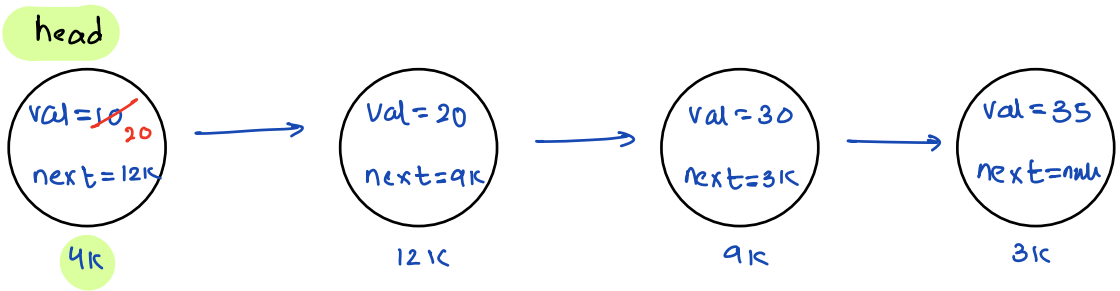
n1.next = n2

4K.next = 8K

* Every node is containing its data and address of the next node.



head = 4K



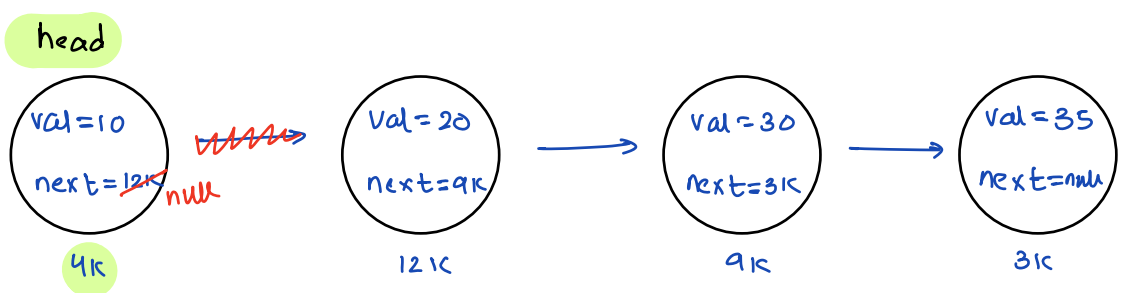
head = 4K

temp = 4K

Node temp = head;

temp.val = temp.next.val;

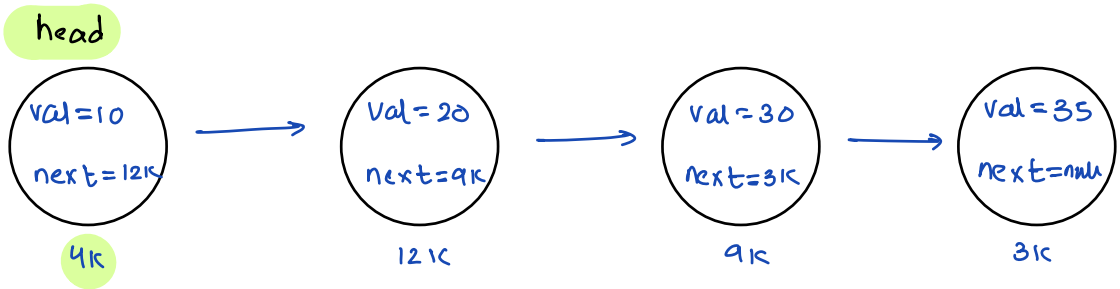
4K.val 20



head = 4K

head.next = null;

4K.next = null



head = 4K

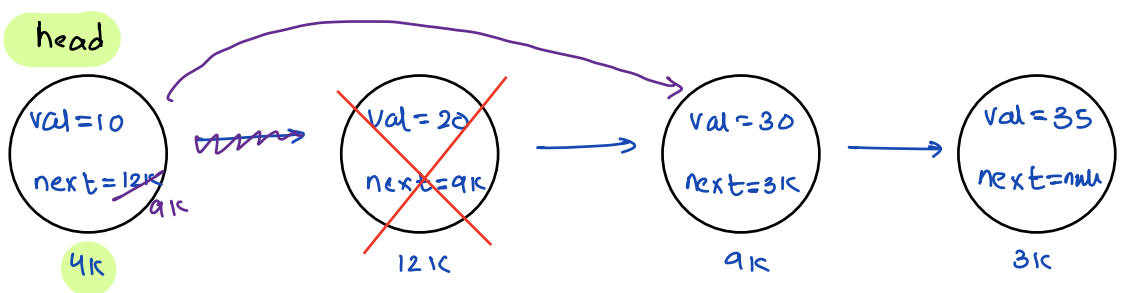
Node temp = head;

temp = temp.next.next;

9K

temp = ~~4K~~ 9K

no changes in LL.



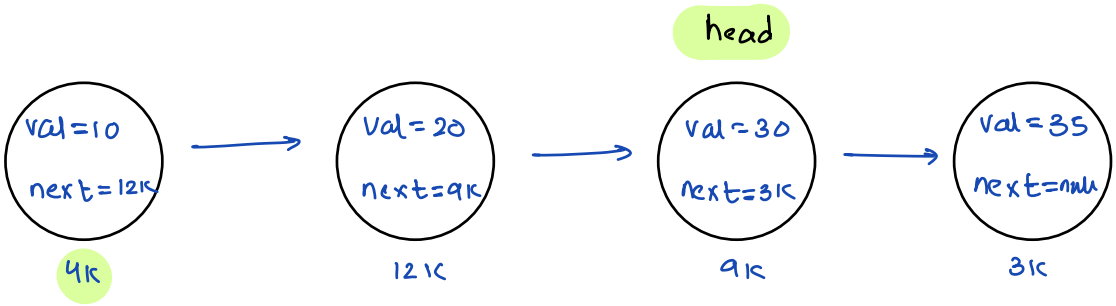
head = 4K

temp = 4K

Node temp = head;

temp.next = temp.next.next;

4K.next = 9K



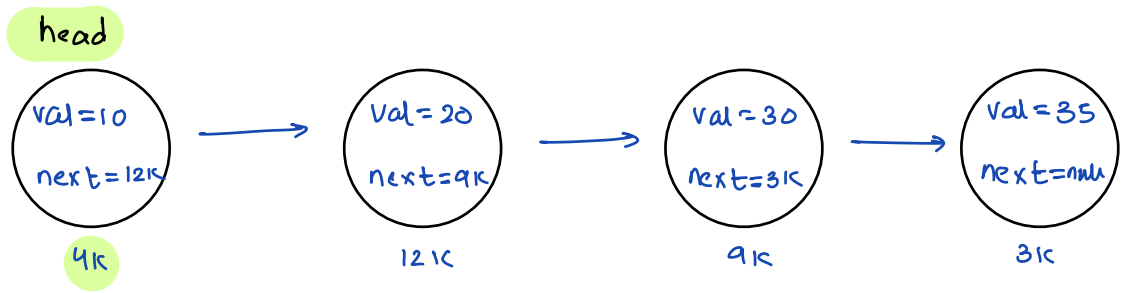

```
head = 4K ;

head = head.next.next;
```

9K

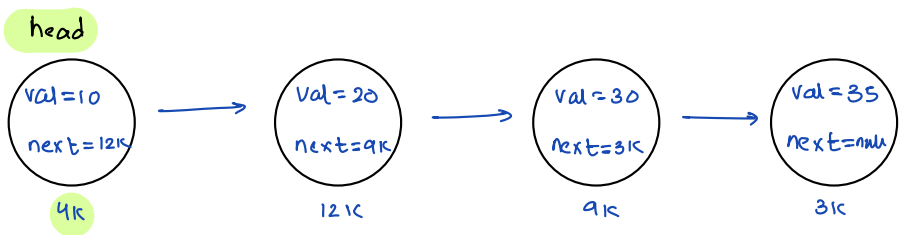
* Don't change the head of LL given to you (unless it is required)

Q.1 Given head of a LL, print its content.



head = 4K

O/P : 10 20 30 35



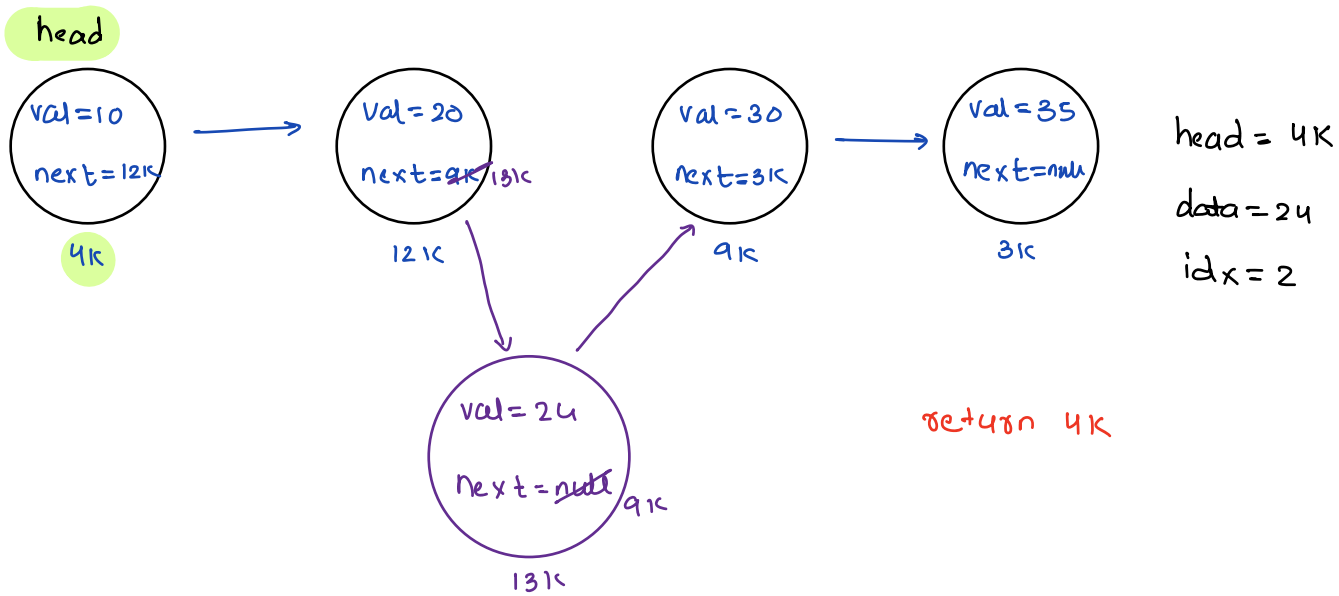

```
void printLL(Node head) {
    Node temp = head;
    while (temp != null) {
        Sopln(temp.val);
        temp = temp.next;
    }
}
```

temp	O/P
4K	10
12K	20
9K	30
3K	35
null	

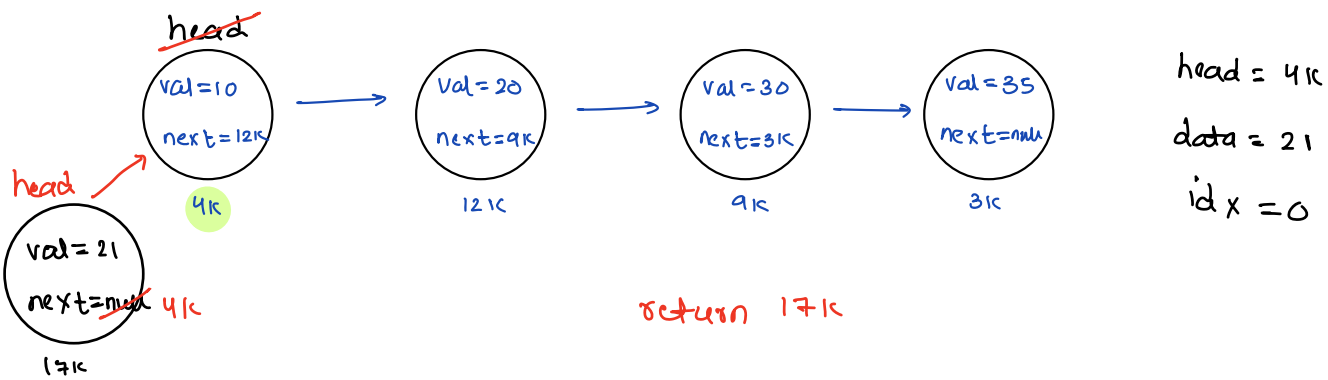
Q.2 **Add node at a particular index in given LL.**

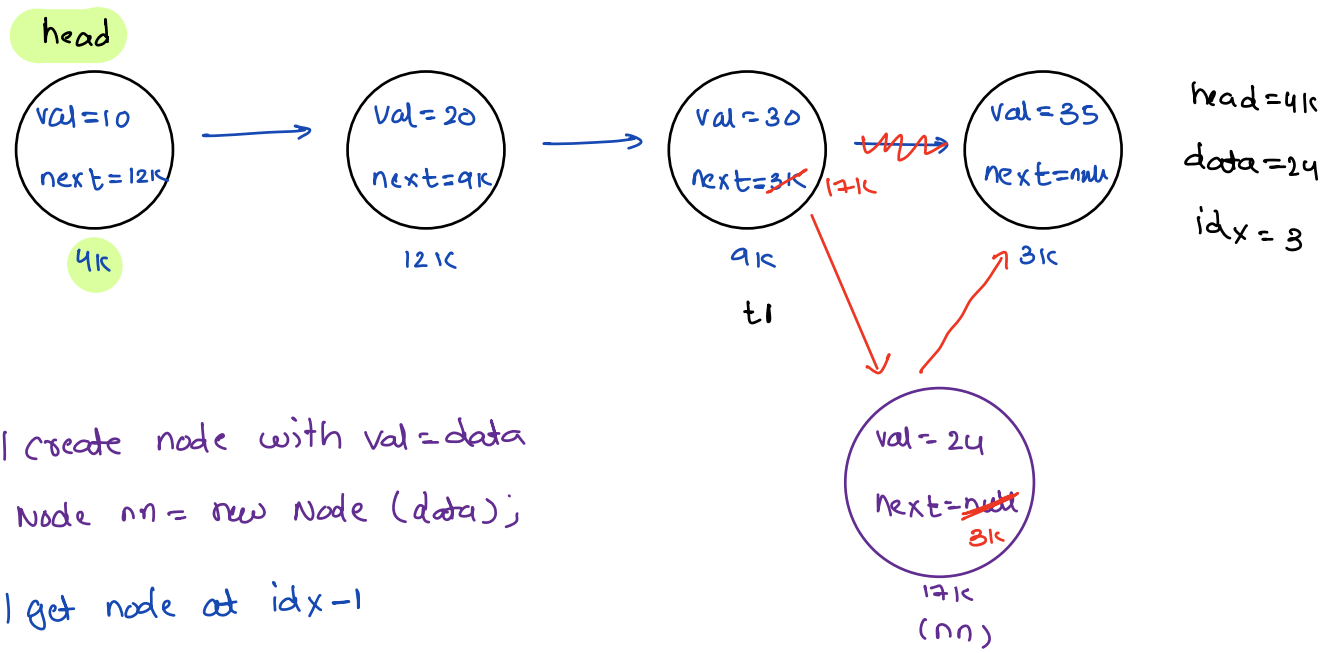
Input: head [head of given LL]

data, idx → create a new node with val = data and add it at index = idx in given LL. Finally return head of LL.



edge case, idx = 0



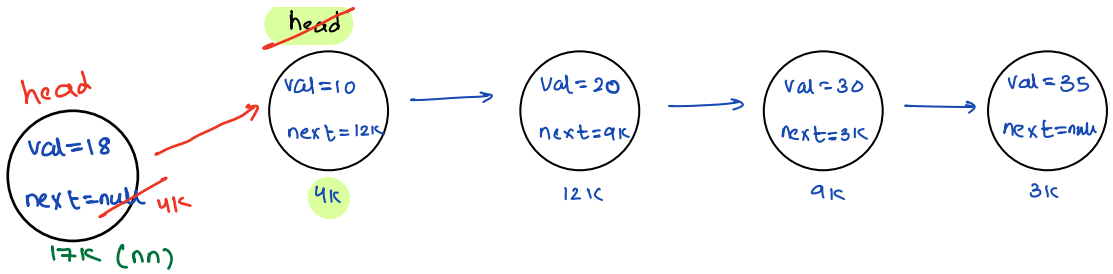



```
// create node with val = data
Node nn = new Node(data);

// get node at idx-1
Node t1 = ....

nn.next = t1.next;
t1.next = nn;
```

```
Node add (Node head, int data, int idx) {
    Node nn = new Node(data);
    if(idx == 0) {
        nn.next = head;
        return nn;
    }
    else {
        //get the node at idx-1
        Node tl = head;
        for (int k = 1; k <= idx-1; k++) {
            tl = tl.next;
        }
        nn.next = tl.next;
        tl.next = nn;
        return head;
    }
}
```

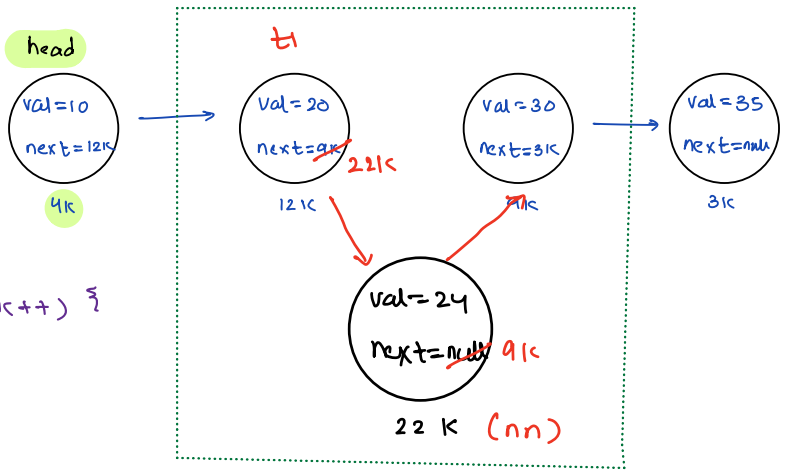



```
Node add (Node head, int data, int idx) {
    Node nn = new Node(data);
    if(idx == 0) {
        nn.next = head;
        return nn;
    }
    else {
        //get the node at idx-1
        Node tl = head;
        for(int k=1; k<=idx-1; k++) {
            tl = tl.next;
        }
        nn.next = tl.next;
        tl.next = nn;
        return head;
    }
}
```

eg1

```
head = 41K
data = 18
idx = 0
```

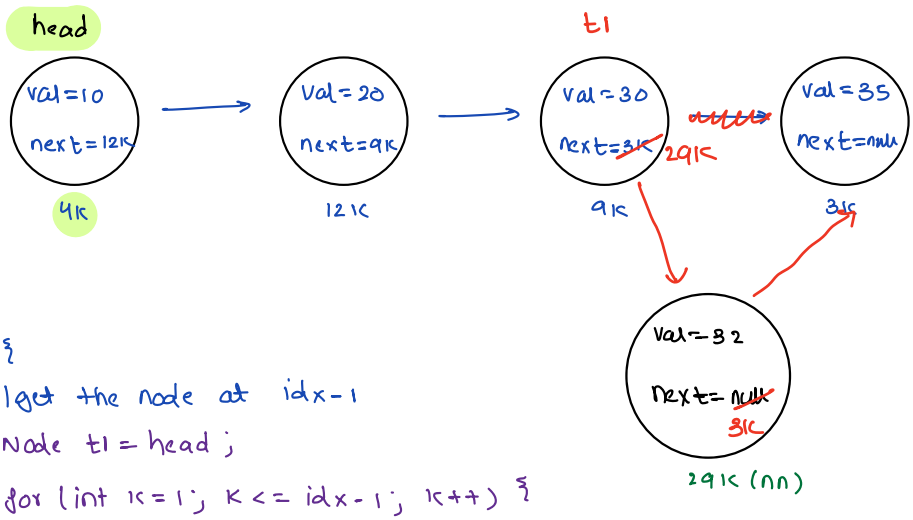
return 17K



eg2

```
head = 41K
data = 24
idx = 2
```

return 4K

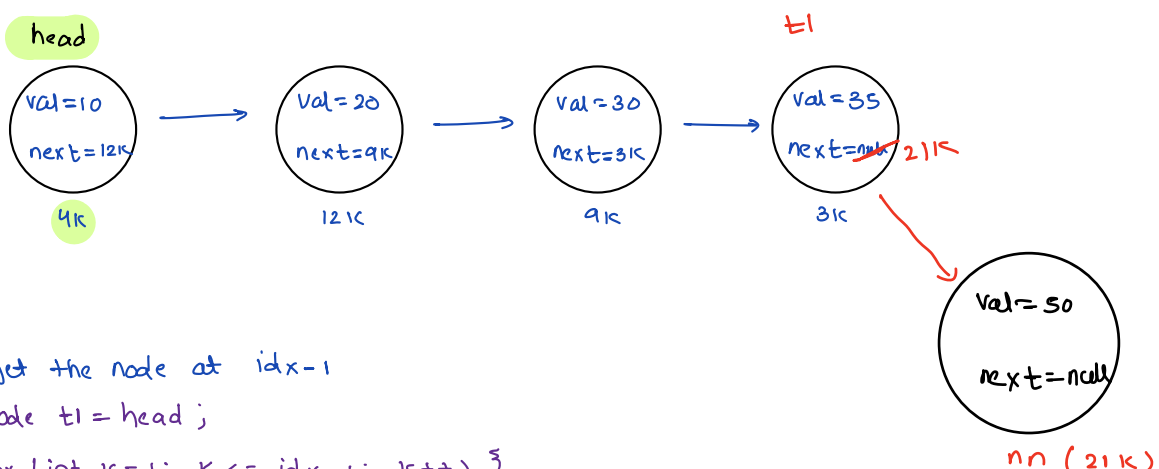



```
else {
    //get the node at idx-1
    Node tl = head;
    for(int i=1; i<=idx-1; i++) {
        tl = tl.next;
    }
    nn.next = tl.next;
    tl.next = nn;
    return head;
}
```

head = 4K

data = 32

idx = 3




```
else {
    //get the node at idx-1
    Node tl = head;
    for(int i=1; i<=idx-1; i++) {
        tl = tl.next;
    }
    nn.next = tl.next;
    tl.next = nn;
    return head;
}
```

head = 4K

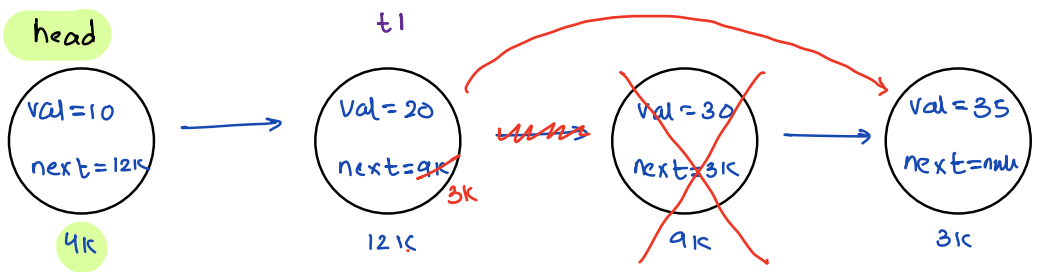
data = 50

idx = 4

Q-3 Delete node from a particular idx.

input : head (head of given LL)

idx (remove node from this index and return head of the LL)



head = 4K

idx = 2

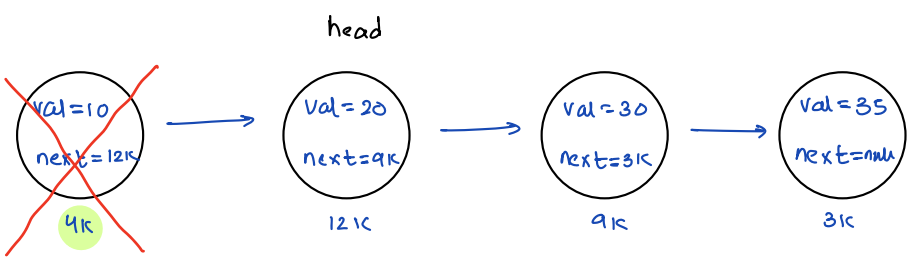
//go to node at idx-1

Node tl = ...

tl.next = tl.next.next

return 4K

Edge case, idx = 0



head = 4K

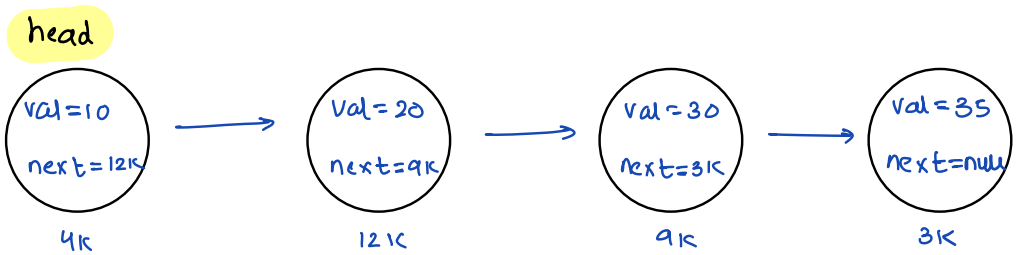
idx = 0

```
Node delete ( Node head, int idx ) {
    if(idx == 0) {
        head = head.next;
        return head;
    }
    else {
        //get the node at idx-1
        Node t1 = head;
        for (int k=1; k<= idx-1; k++) {
            t1 = t1.next;
        }
        t1.next = t1.next.next;
        return head;
    }
}
```

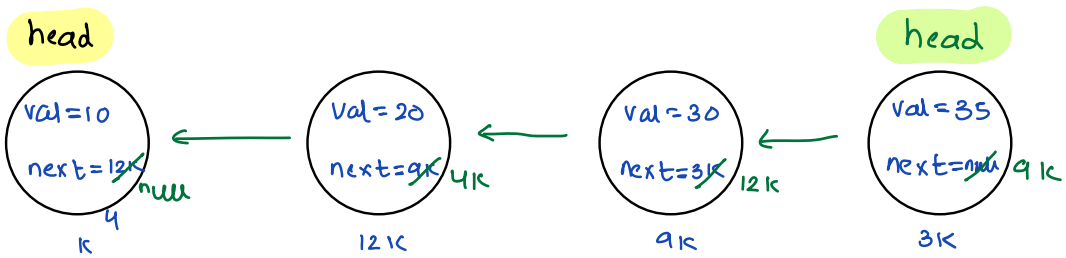
→ Diff b/w LL and Arrays

	LL	Arrays
memory	discontous memory allocation	continous memory allocation
access	travel is required ~ O(N)	A[idx] => O(1)
add / delction	its easy to manage	very difficult to manage

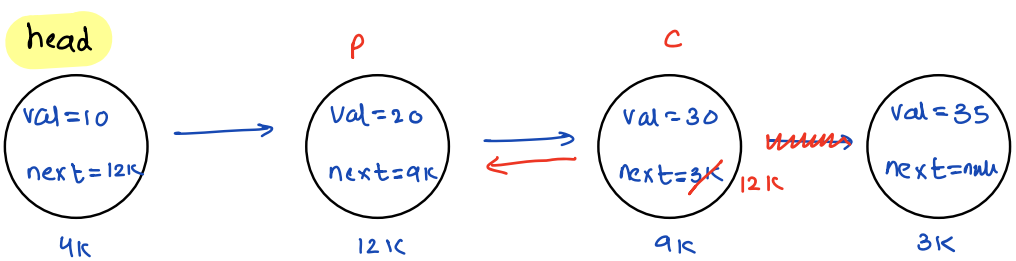
Q.4 Given head of LL. Reverse LL and return reversed LL head.



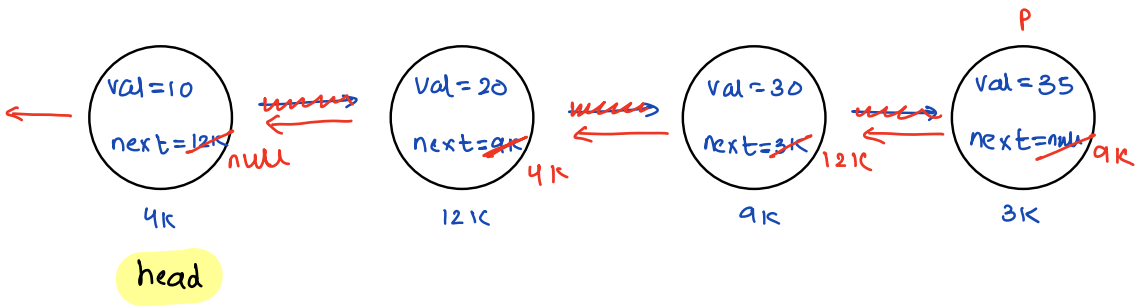
after reverse



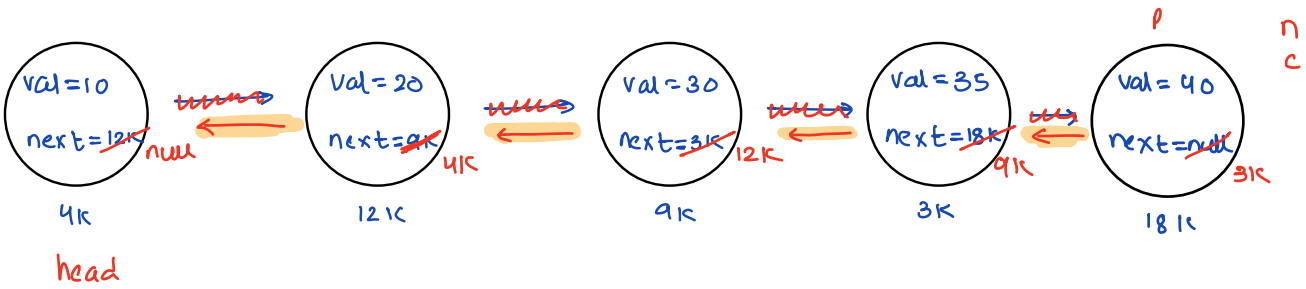
hint



c.next = p




```
Node reverseLL(Node head) {
    Node p = null, c = head;
    while (c != null) {
        Node n = c.next;
        c.next = p;
        p = c;
        c = n;
    }
    return p;
}
```




```
Node reverseLL(Node head){
    Node p = null, c = head;
    while(c != null){
        Node n = c.next;
        c.next = p;
        p = c;
        c = n;
    }
    return p;
}
```

return 18K