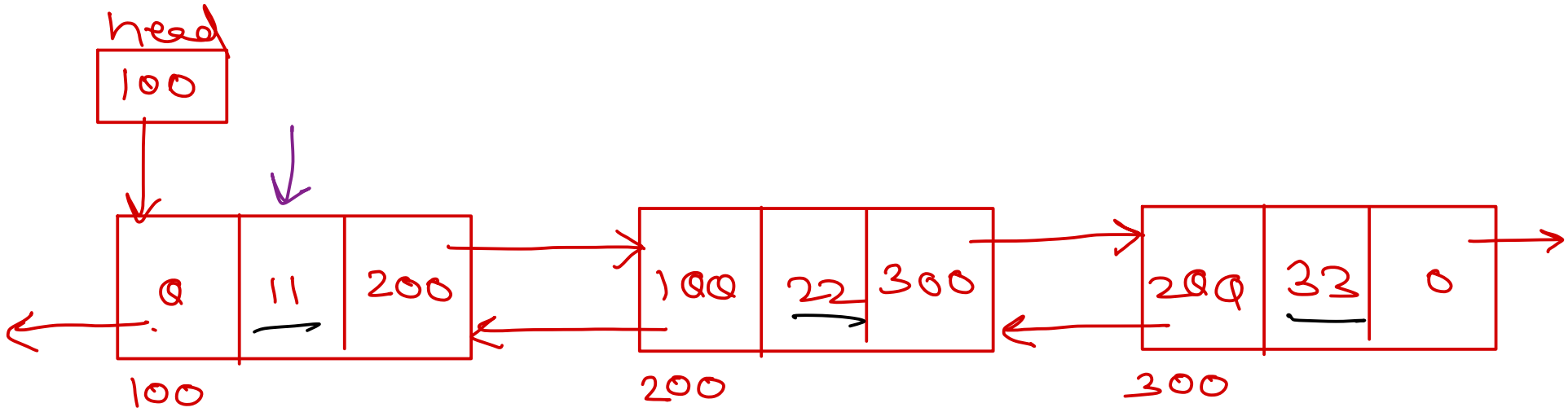
Data Structure & Algorithms

Nilesh Ghule

Doubly Linear Linked List → bidirectional traversal. – displayFwd() displayRev()

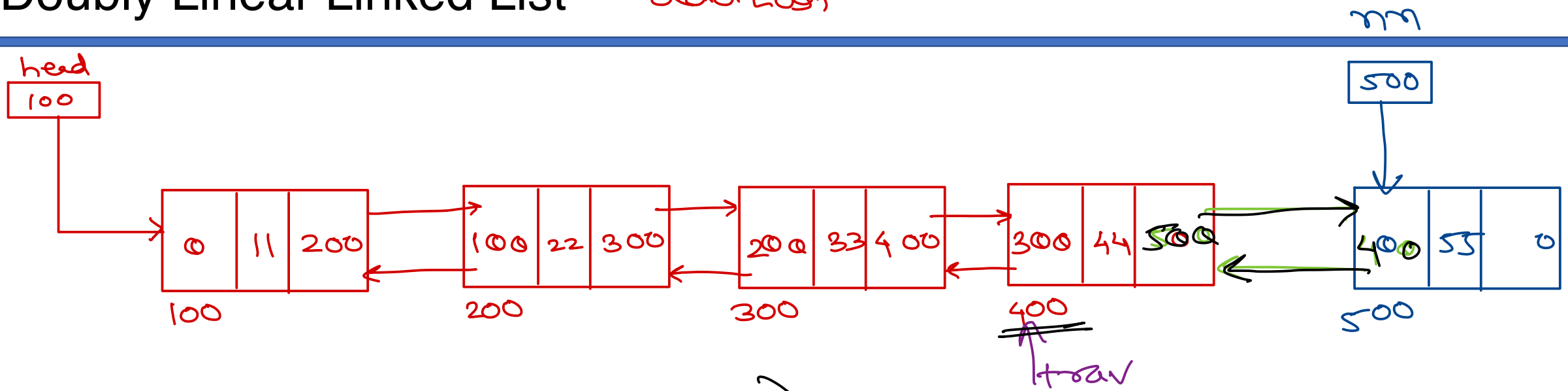



```
trav = head;
while (trav != null) {
    print (trav.data);
    trav = trav.next;
}
```

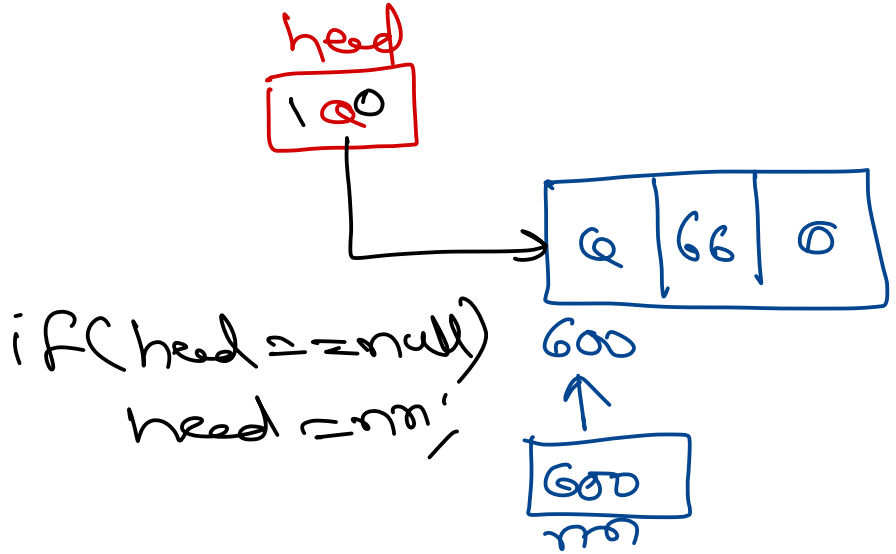
①
```
trav = head;
while(trav.next != null)
    trav = trav.next;
```

②
```
while (trav != null)
{   print (trav.data);
    trav = trav.prev;
}
```

Doubly Linear Linked List – addLast

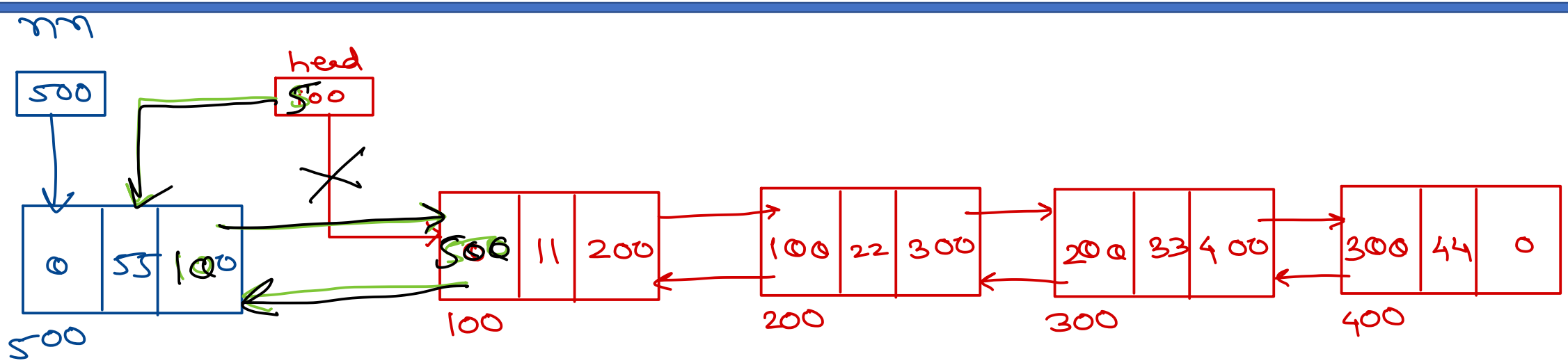



```
① Node nn = new Node(val);
② Node trav = head;
   while (trav.next != null)
       trav = trav.next;
③ trav.next = nn;
④ nn.prev = trav;
```




```
if(head==null)
    head=nn;
```

Doubly Linear Linked List - addFirst()



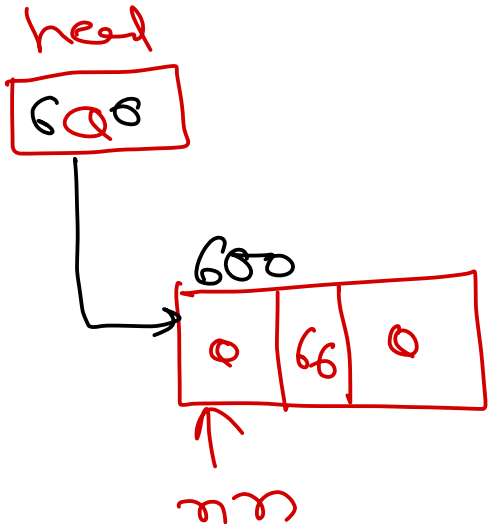
① Node nn = new Node(val);

② nn.next = head;

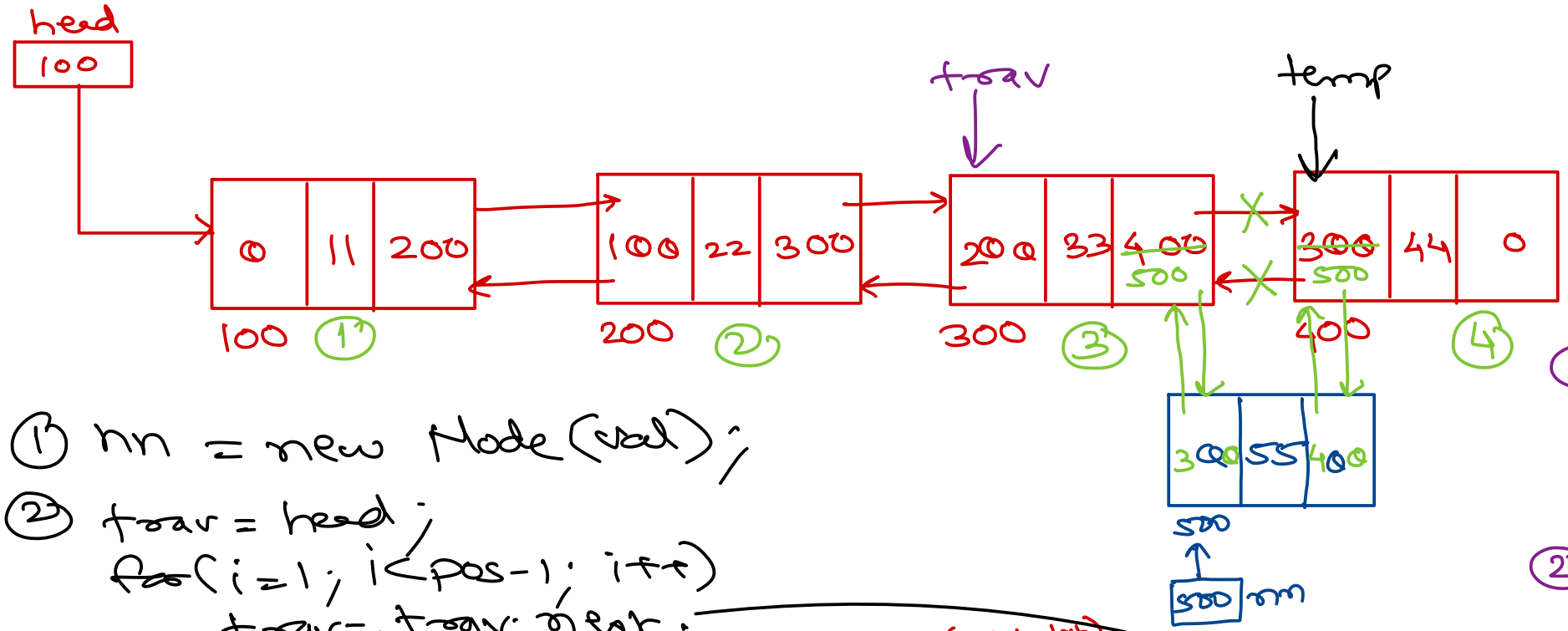
③ head.prev = nn;

④ head = nn;

```
if(head==null)
    head=nn;
```



Doubly Linear Linked List – addAtPos()



(1) nn = new Node(val);

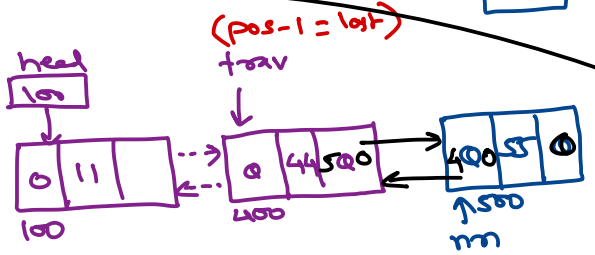
(2) trav = head;
for(i=1; i<pos-1; i++)
trav = trav.next;

(3) temp = trav.next;

(4) nn.next = temp;
nn.prev = trav; X

(5) trav.next = nn;

(6) temp.prev = nn; X

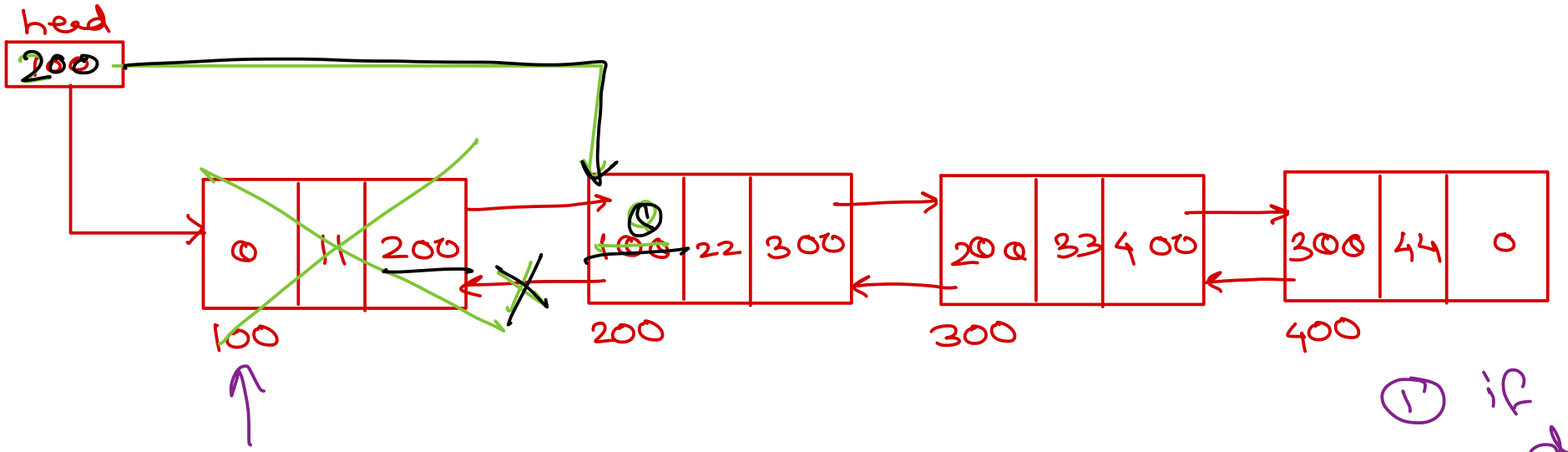


(1) if list empty or pos == 1,
addFirst();

(2) if adding node at last pos, temp will be null.
if(temp != null)
temp.prev = nn;

(3) if adding node beyond last pos, add node at the end.
if(trav.next == null)
break;

Doubly Linear Linked List - delFirst()



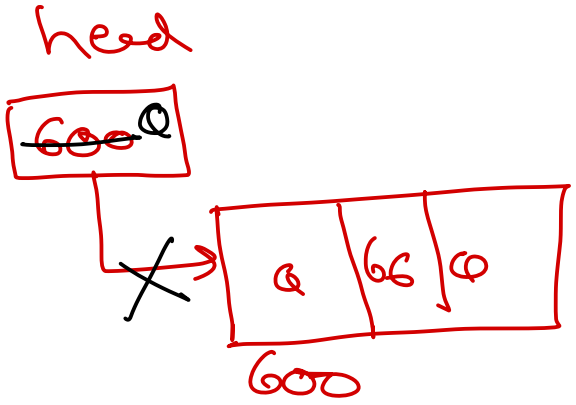
① head = head.next;

② head.prev = null;

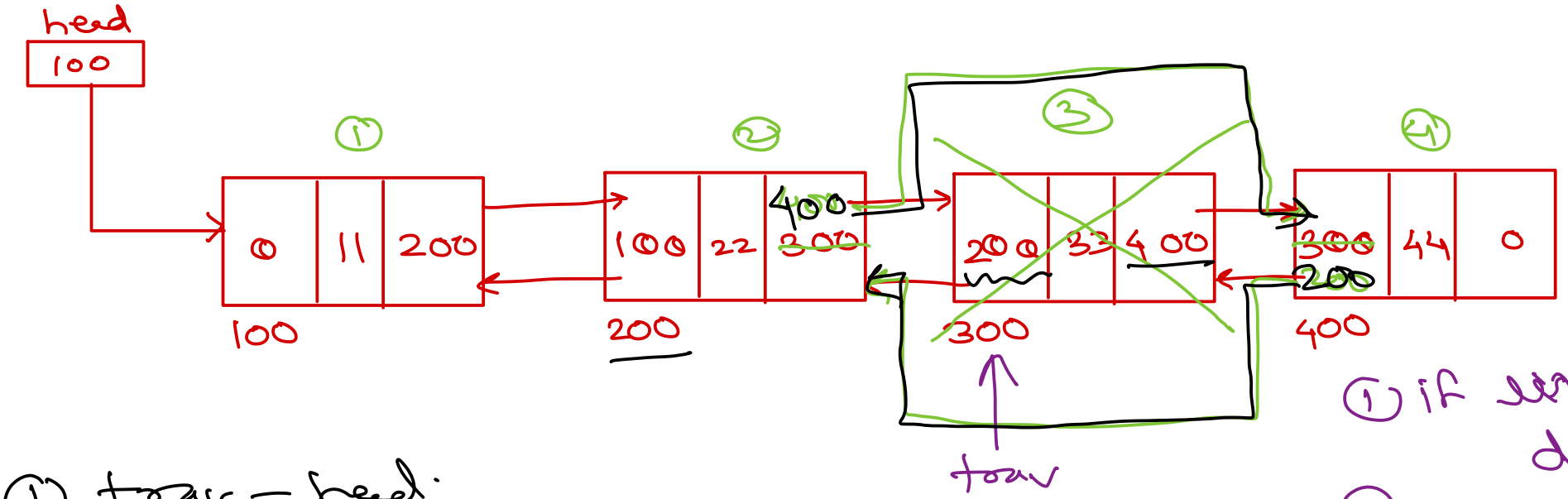
① if list is empty, do nothing.

② if list has single node,

```
if(head.next==null)
    head = null;
```



Doubly Linear Linked List - delAtPos(3)



① if list empty or pos <=1, delFirst();

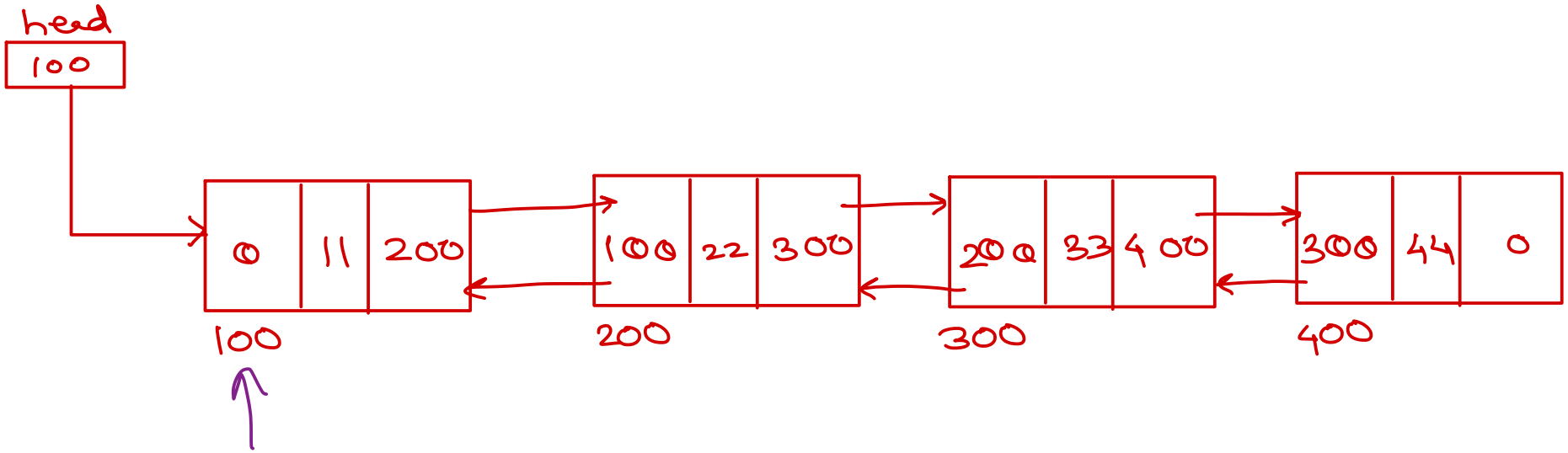
② if deleting beyond the pos, do nothing

③ if deleting on last pos, trav.next will be null.

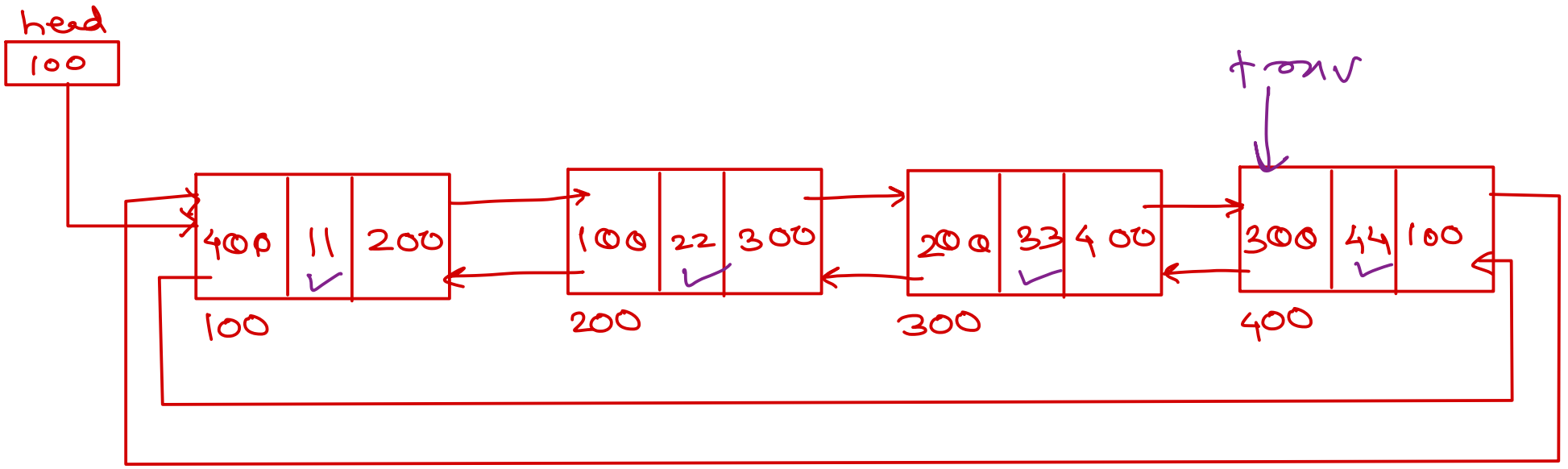
```
① trav = head;
   for(i=1; i<pos; i++)
       trav = trav.next;

   if(trav==null)
       return;

② trav.prev.next = trav.next;

③ if(trav.next != null)
       trav.next.prev = trav.prev;
```

Doubly Linear Linked List

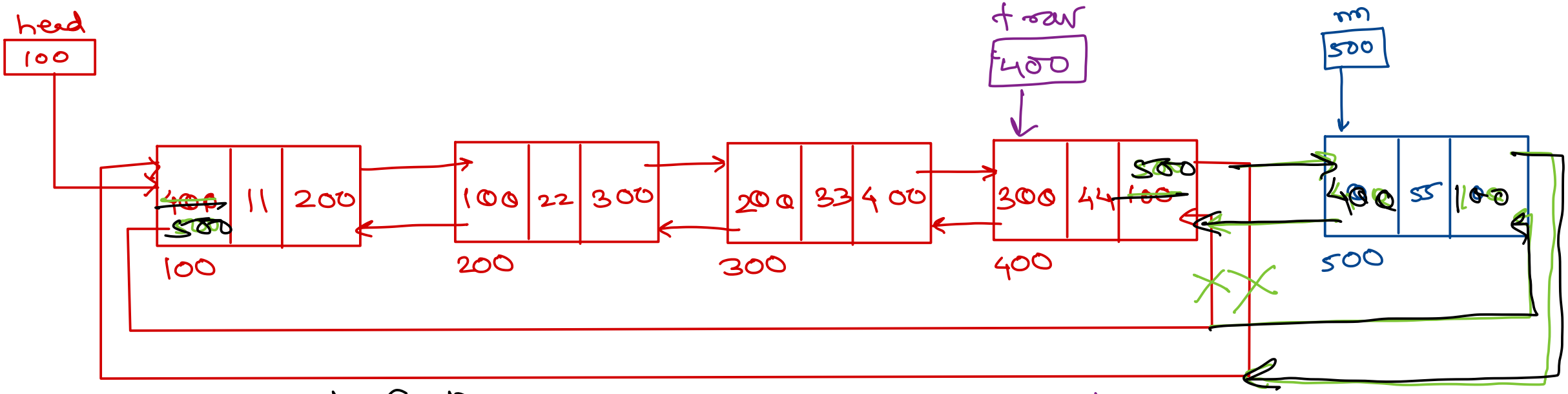
- delLast()
- homework.



Doubly Circular Linked List - display Rev ( )

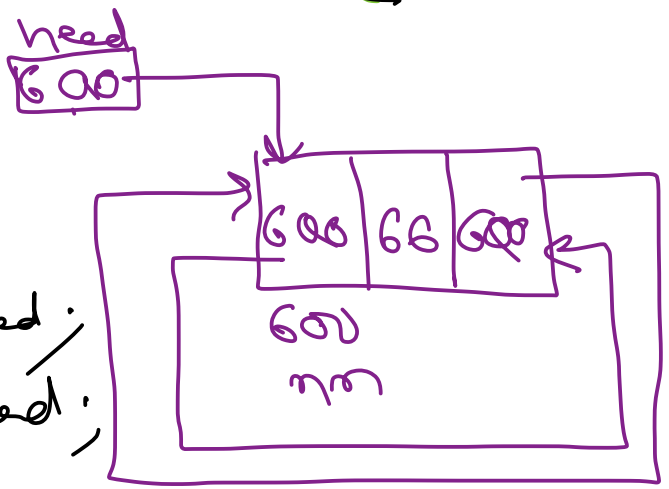


Doubly Circular Linked List – addLast()



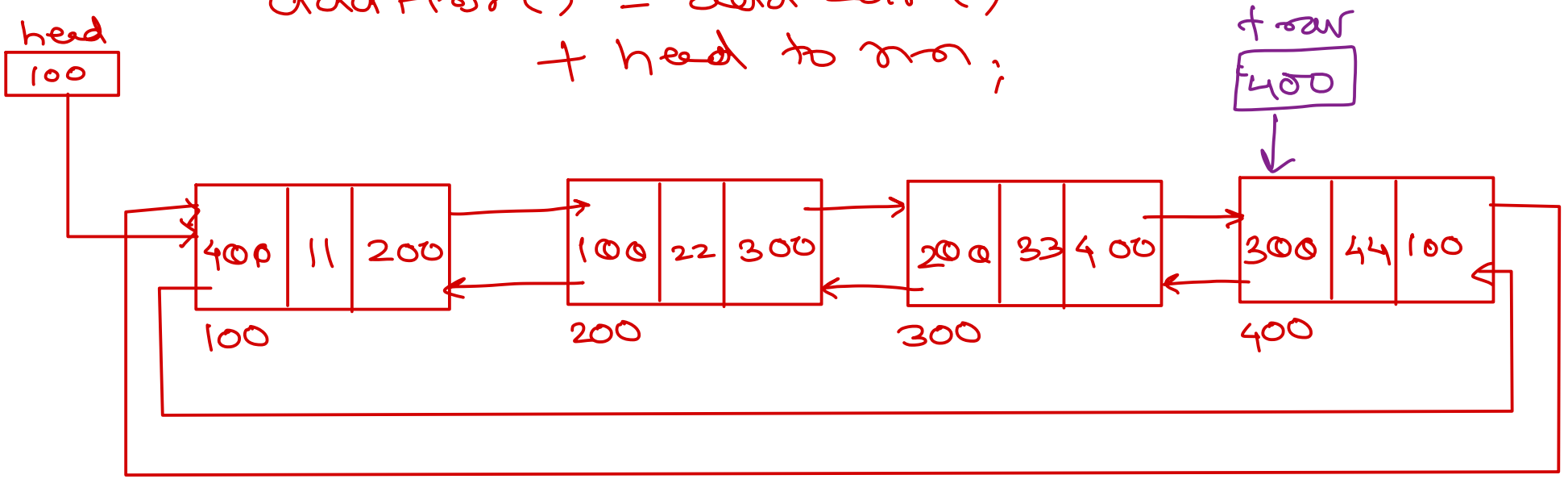

```
nn = new Node(val);
trav = head.prev;
nn.next = head;
nn.prev = trav;
trav.next = nn;
head.prev = nn;
```

```
if (head == null) {
    head = nn;
    nn.next = head;
    nn.prev = head;
}
```



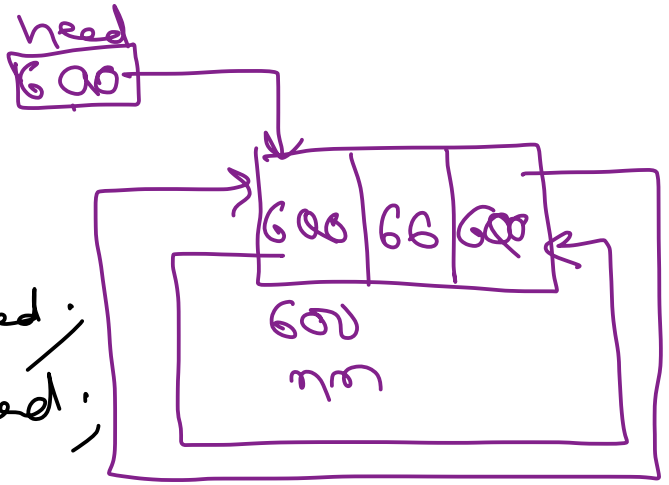
Doubly Circular Linked List – addFirst()

addFirst() = addLast() + head to nn;

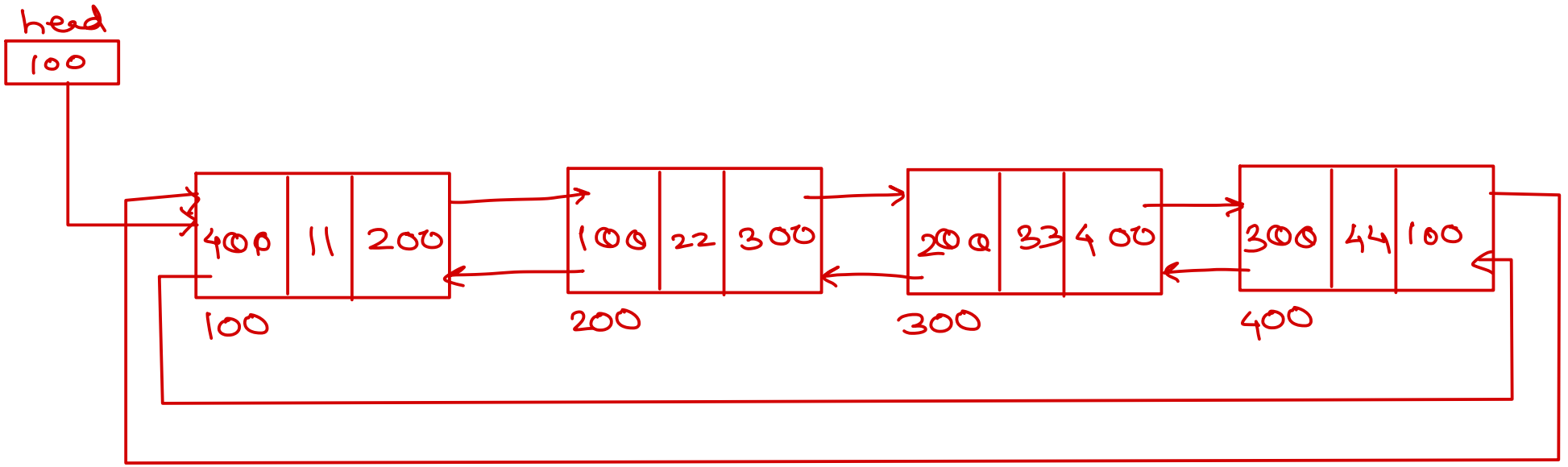



```
nn = new Node(val);
trav = head.prev;
nn.next = head;
nn.prev = trav;
trav.next = nn;
head.prev = nn;
head = nn;
```

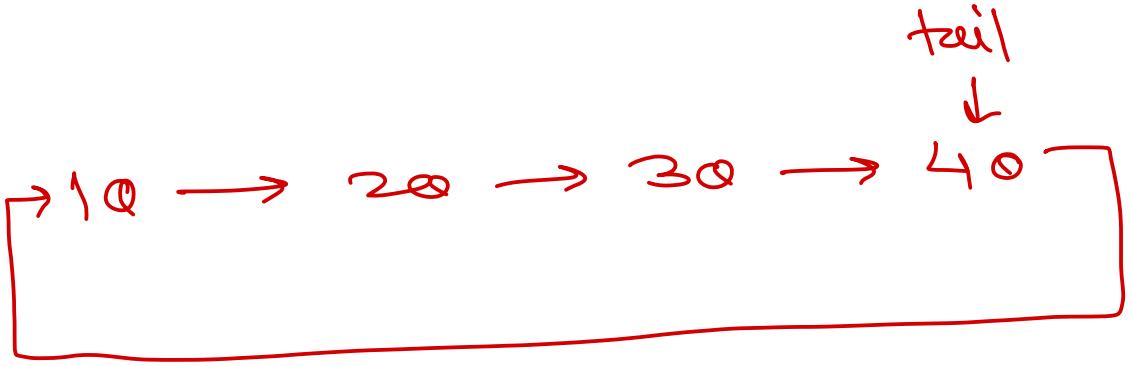
```
if (head == null) {
    head = nn;
    nn.next = head;
    nn.prev = head;
}
```

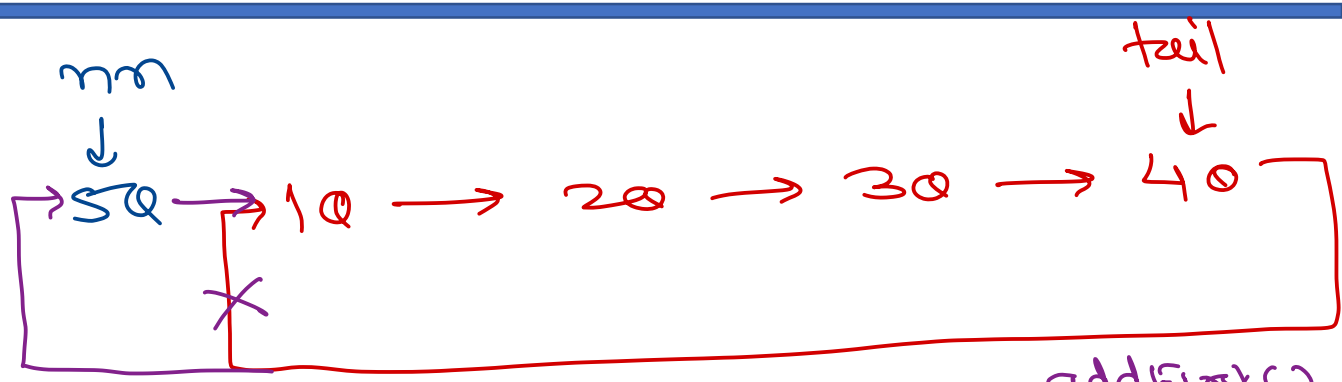


Doubly Circular Linked List



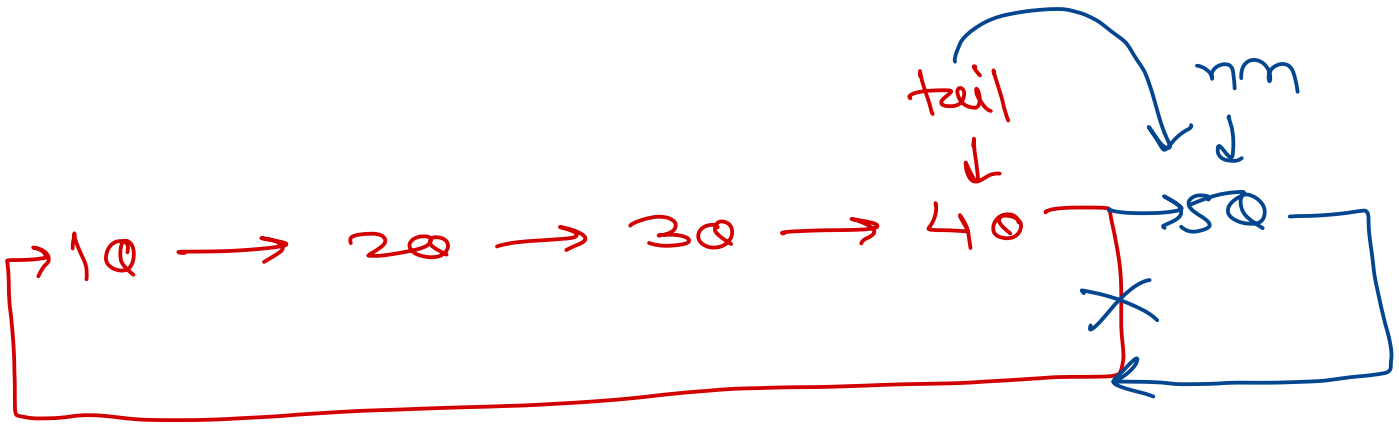
Singly Circular Linked List – with tail pointer





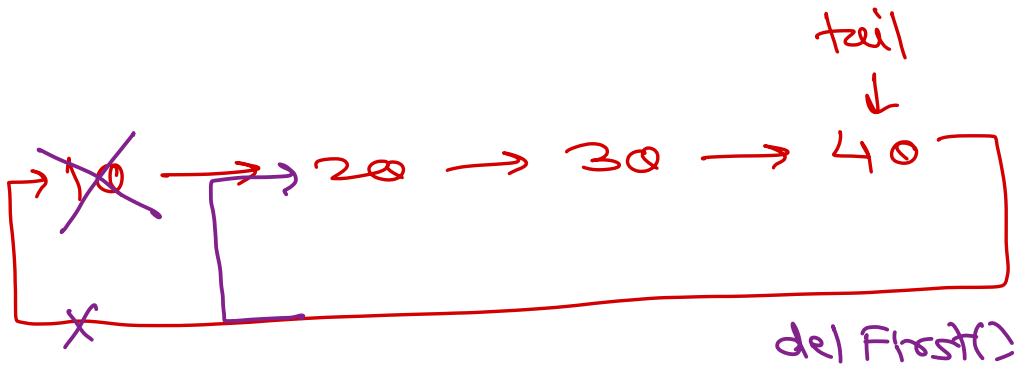
nn.next = tail.next;
tail.next = nn;

addFirst()
O(1)



nn.next = tail.next;
tail.next = nn;
tail = nn;

addLast()
O(1)

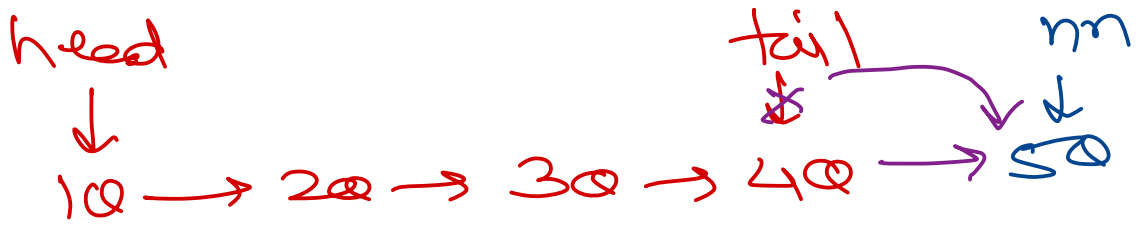


tail.next = tail.next.next;

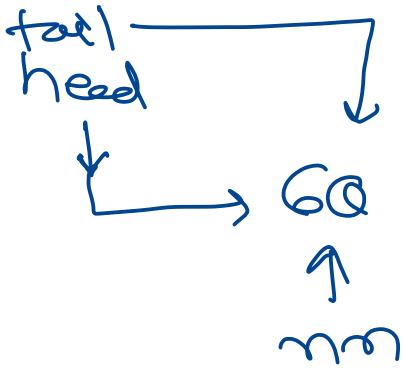
delFirst()
O(1)

Singly Linear Linked List - with head & tail pointer

addLast()

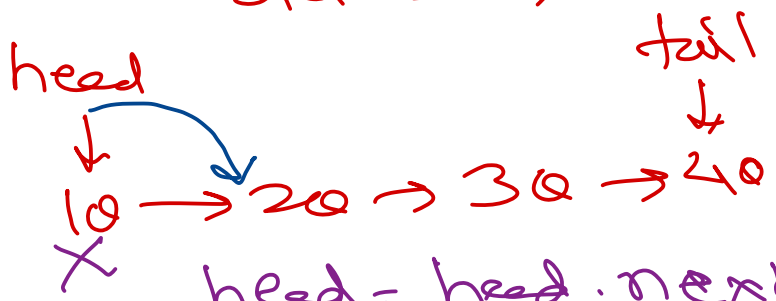



```
tail.next = nn;
tail = nn;
```




```
if(head==null) {
    head=nn;
    tail=nn;
}
```

delLast()




```
head = head.next;
```

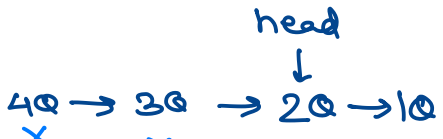

```
if(head.next==null){
    head=null;
    tail=null;
}
```

Stack / Queue using Linked List

- Stack can be implemented using linked list. → LIFO.
  - add first → push() → O(1)
  - delete first → pop() → O(1)
  - is empty → head == null.

In stack addition & deletion both are done from same end.

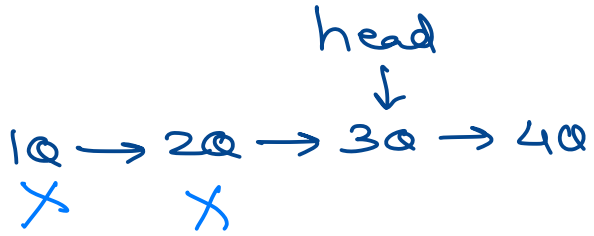
push	pop
10	40
20	30
30	↓
40	



- Queue can be implemented using linked list. → FIFO
  - add last → push() → O(1)
  - delete first → pop() → O(1)
  - is empty → head == null.

In queue, addition & deletion done from different ends.

push	pop
10	10
20	20
30	↓
40	



Linked List – Competitive programming

- Sort the singly linked list. → selection sort

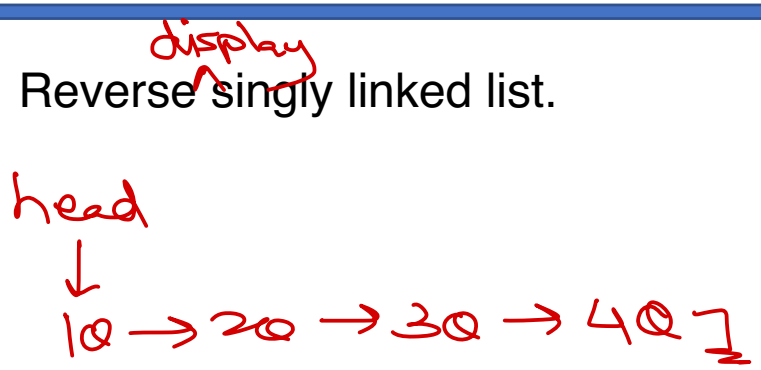
```
Node i, j;
for(i=head; i!=null; i=i.next) {
    for(j=i.next; j!=null; j=j.next) {
        if(i.data > j.data) {
            temp = i.data;
            i.data = j.data;
            j.data = temp;
```

3
3
3

bubble sort
– homework

Linked List – Competitive programming

- Reverse singly linked list.



way 1:

① put all elements on stack one by one

② pop all elements from stack & print them.

T ∝ 2n → O(n)

space →
linked list → O(n)
stack → O(n)

```
① S = new Stack<>
   trav = head;
   while (trav != null) {
       S.push(trav.data);
       trav = trav.next;
   }

② while (!S.isEmpty()) {
       val = S.pop();
       print(val);
   }
```

Linked List – Competitive programming

- Reverse singly linked list.

display

Time = $O(n^2)$

head

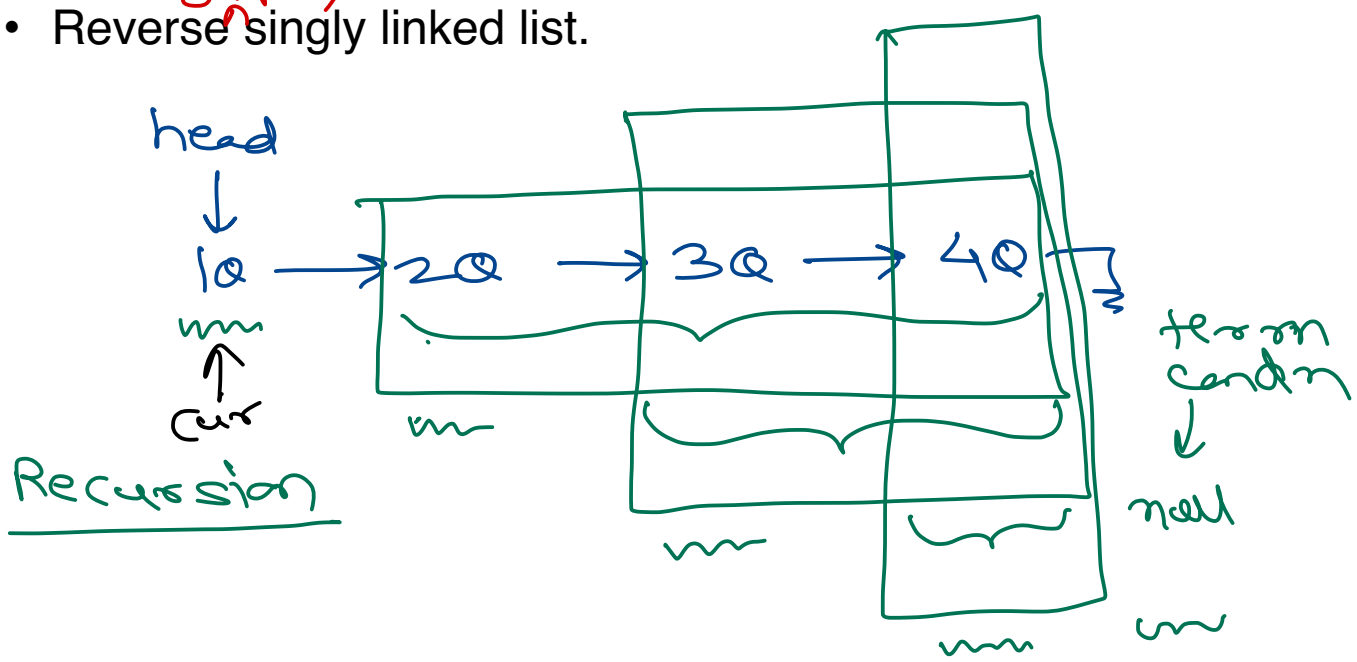
1. traverse list & count node = n;
2. traverse till n deg & print last element.
3. decrement n;
4. repeat steps 2 & 3 until n = 0.

Space: list $O(n)$

aux space = $O(1)$

Linked List – Competitive programming

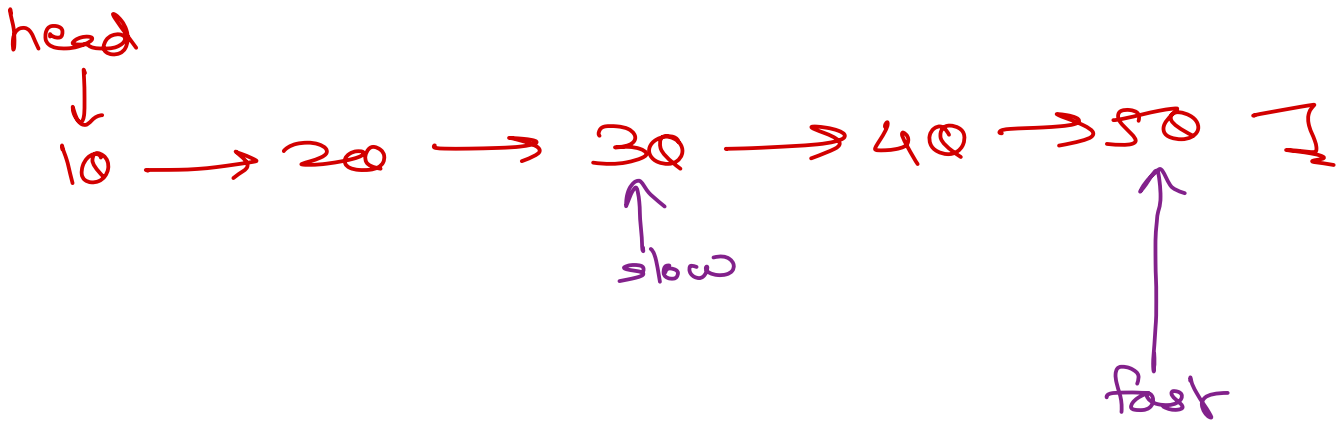
- Reverse singly linked list.




```
void revDisplay(Node cur) {
    if(cur==null)
        return;
    revDisplay(cur.next);
    print(cur.data);
}

revDisplay(head);
```

Linked List – Competitive programming

- Find middle of singly linear linked list.



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>