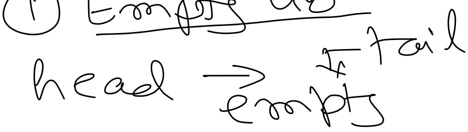
Insert in a doubly linked list

```
Insert(element) - with no tail pointer.
- Make memory for new element, say newNode.
- Store element in newNode's data.
- Set newNode's next and previous to empty.
- if list is empty then
  - Set head and tail to newNode.
  - Stop.
// List is not empty. Traverse the list to find ~~previous and~~ current nodes,
because newNode will be inserted ~~between previous and~~ before current
nodes..
- Set current to head.
- ~~Set previous to empty.~~
- while ( (current is not empty) and (current node's data < element) )
  - ~~Set previous to current.~~
  - Set current to current's next.
// Adding before first node? (Adding smallest element).
- if current is head node then
  - Set newNode's next to head.
  - Set head's previous to newNode.
  - Set head to newNode.
  - Stop.
// Adding after last node? (Adding largest element).
- if current is empty then
  - Set tail's next to newNode. // After last node comes newNode.
  - Set newNode's previous to tail. // Before newNode comes last node.
  - Set tail to newNode.
  - Stop.
- ~~Set previous node's next to newNode~~.
- Set (current node's previous) node's next to newNode.
- Set neNode's previous to current node's previous.
- Set newNode's next to current.
- Set current node's previous to newNode.
```

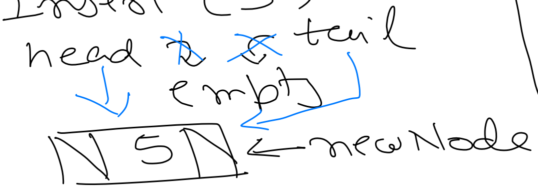
To be done if list has tail pointer

Text scratched out & underlined above in RED color are difference between singly & doubly list algo.

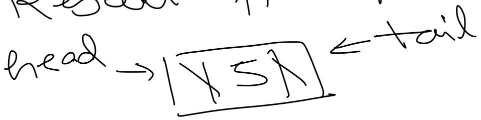
① Empty list



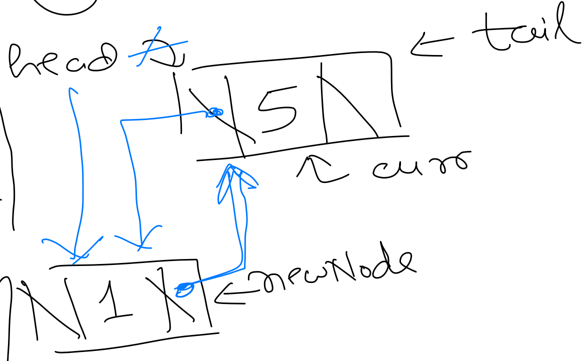
② Insert (5)



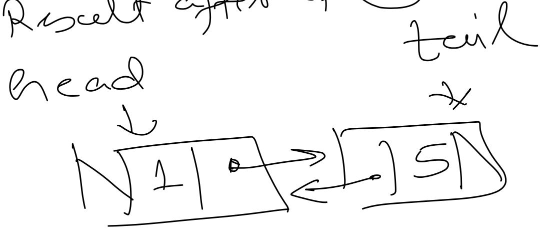
Result after op ②



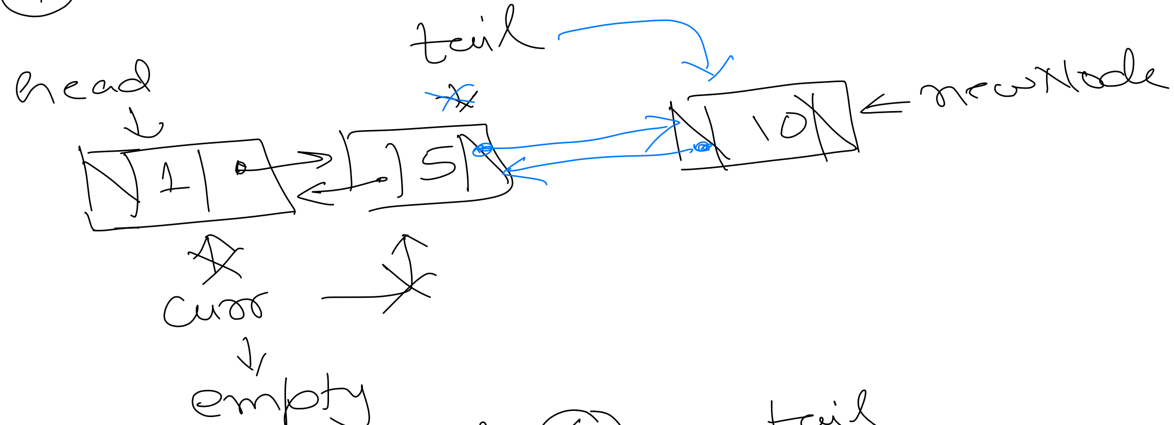
③ Insert (1)



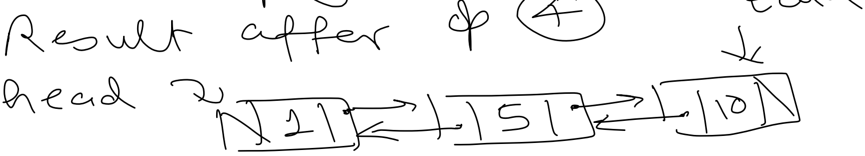
Result after op ③



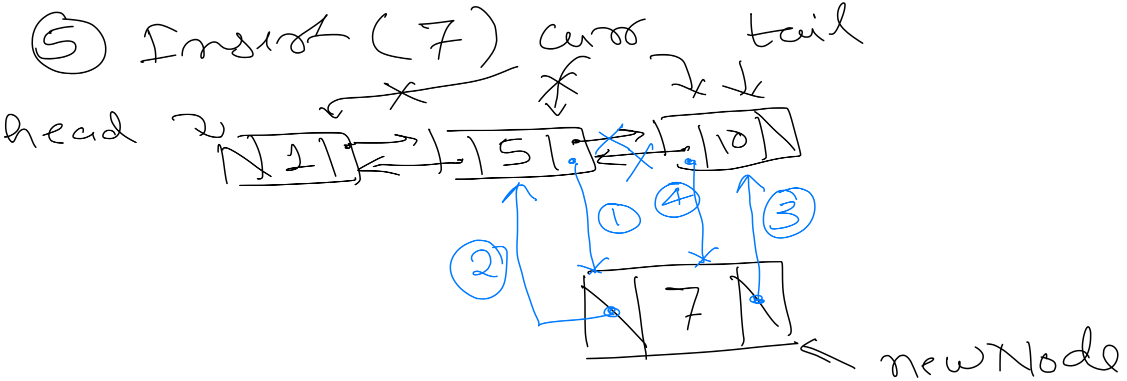
④ Insert (10)

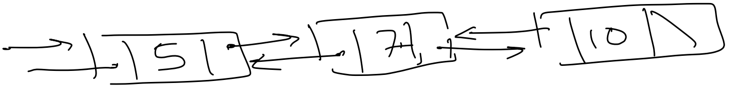


Result after op ④



⑤ Insert (7)





① Set current node's previous node's next to newNode

② Set newNode's previous to current node's previous.

③ Set newNode's next to curr.

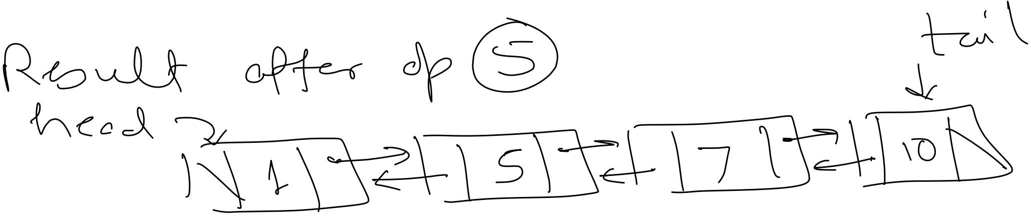
④ Set current's previous to newNode.

Before Step #4, Step # 1 should be done.

which node comes before current node

- Set (current node's previous) node's next to newNode.
- Set neNode's previous to current node's previous.
- Set newNode's next to current.
- Set current node's previous to newNode.

Result after op ⑤

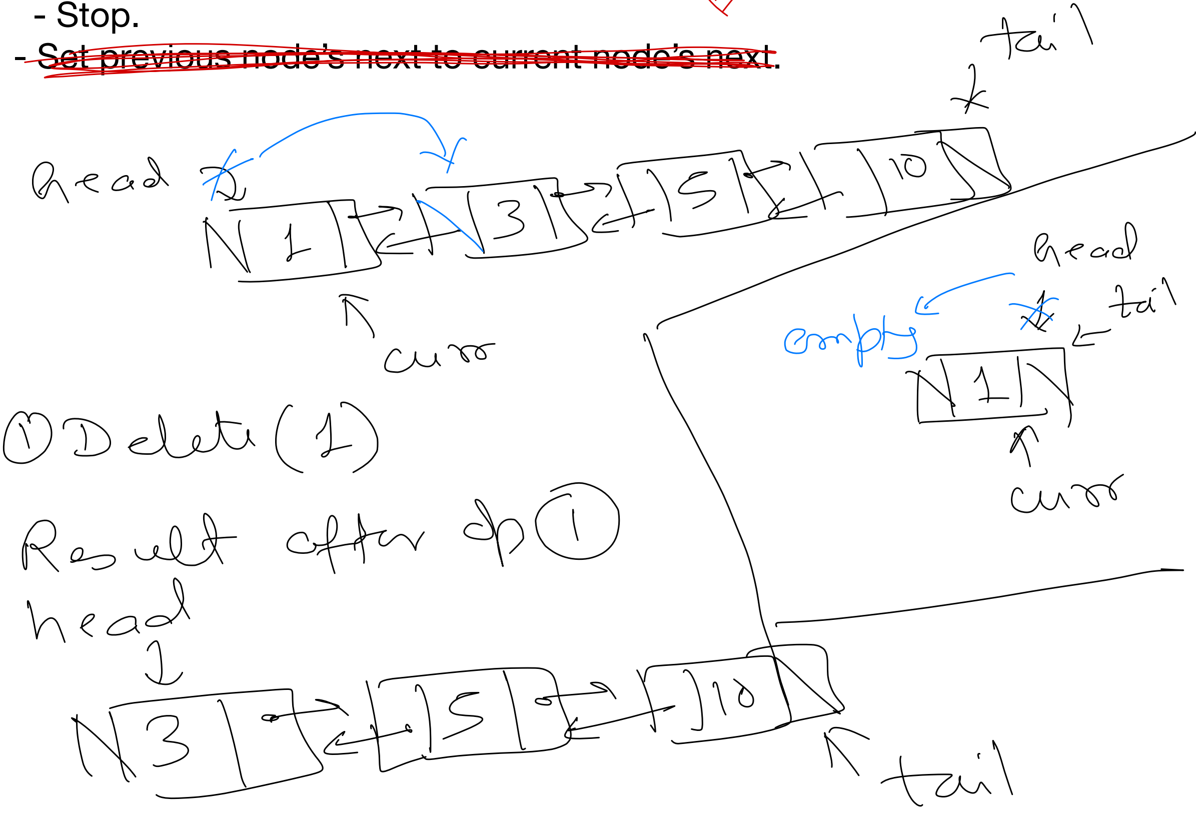


Delete

Singly list algo

```
Delete(element)
- Set current to first node of list.
~~- Set previous to empty.~~
- while (current is not empty) do
   - if current node's data is element then
     - end the loop.
   ~~- Set previous to current node.~~
   - Set current to current node's next.
- if current node is empty then
     // No node to be deleted as element not found OR list is empty.
  - Stop.
- if current node is the first node then
    // Deleting the first node of linked list.
  - Set head to current node's next.
  - Stop.
~~- Set previous node's next to current node's next.~~
```

Changes to be made for doubly list.



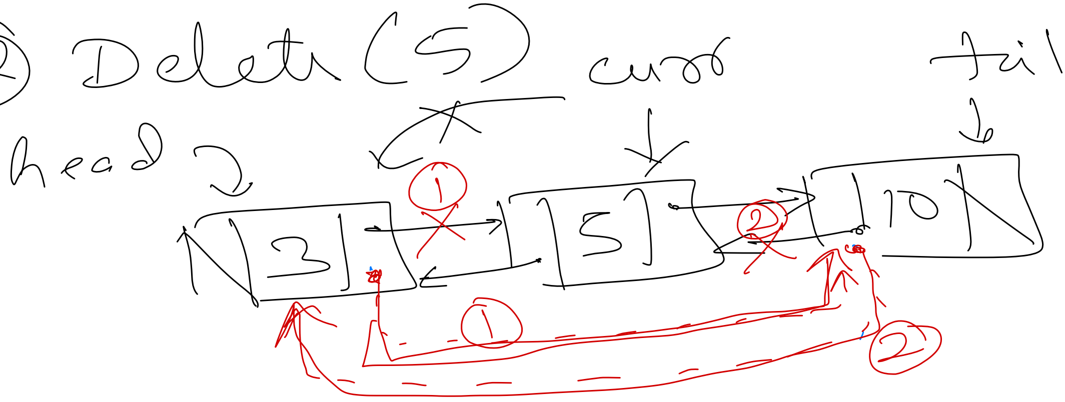
① Delete(1)

Result after op ①

Delete element from doubly list

```
Delete(element)
- Set current to first node of list.
- while (current is not empty) do
    - if current node’s data is element then
      - end the loop.
  - Set current to current node’s next.
- if current node is empty then
      // No node to be deleted as element not found OR list is empty.
  - Stop.
- if current node is the first node then
    // Deleting the first node of linked list.
  - Set head to current node’s next.
  - if head is empty then
    // List had only one node and that we are to delete => List will be empty.
    - Set tail to empty.
    Else
    // List is not empty.
   - Set head node’s previous to empty.
  - Stop.
- if current node is the last node then
  // Deleting the last node of the linked list.
  - Set (current node’s previous) node’s next to empty.
  - Set tail to (current node’s previous).
  - Stop.
- Set (current node’s previous) node’s next to (current node’s next).
- Set (current node’s next) node’s previous to (current node’s previous).
```

② Delete(5)



Node 3

① Set current node's previous node's next to current node's next

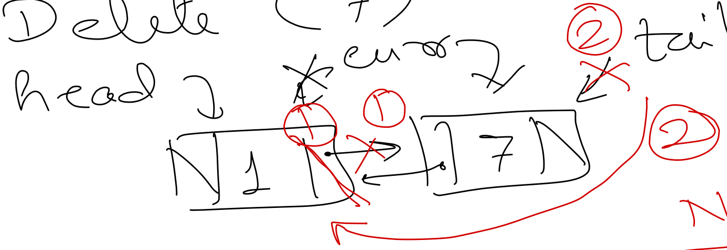
Node 10

Node 10

② Set current node's next node's previous to current node's previous

Node 3

③ Delete (7) => Deleting last node



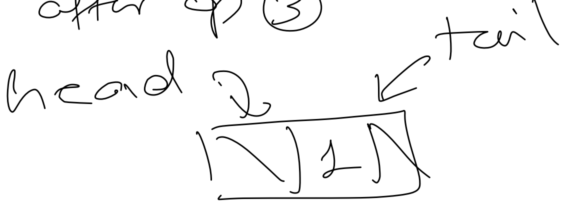
Node 1

① Set curr node's previous node's next to empty.

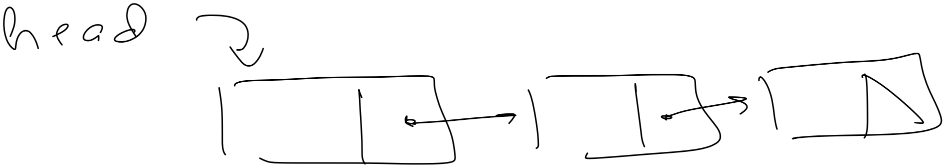
② Set tail to curr node's previous

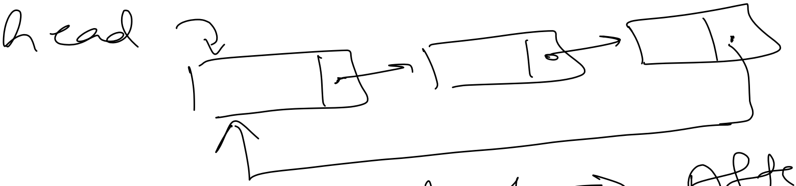
Node 1

Result after step ③

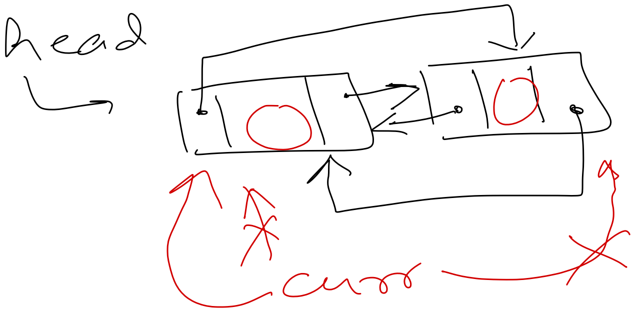


Circular linked list





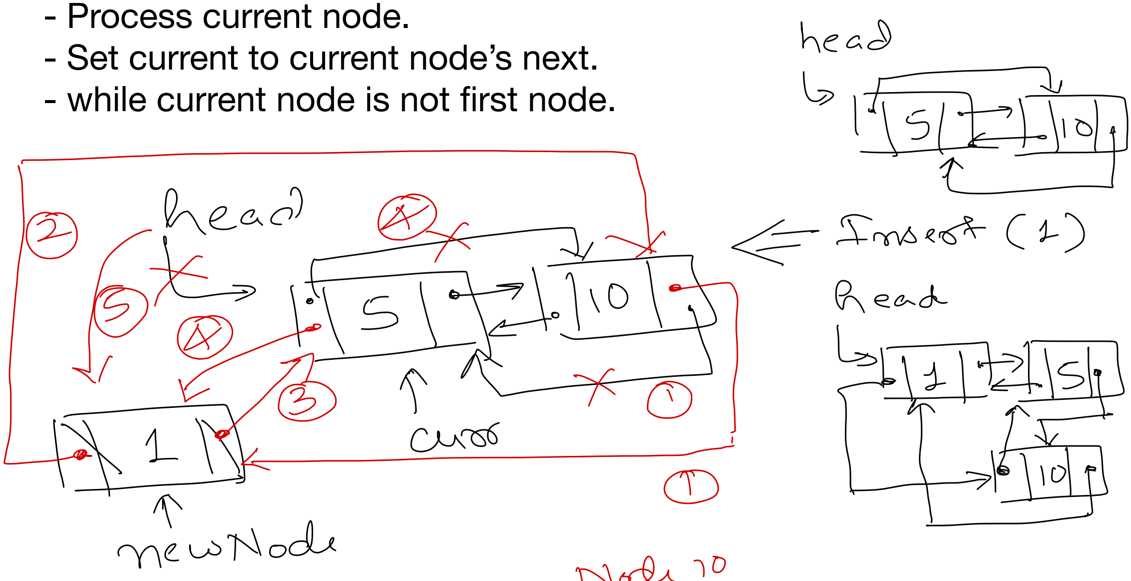
Circular list ⇒ ~~After~~ last node, we can come to first node.



head
↓
empty

Forward Traversal - Circular list – with no dummy head node.

```
- if list is empty then
  - Stop.
- Set current to first node.
- do
  - Process current node.
  - Set current to current node's next.
  - while current node is not first node.
```



Node 10

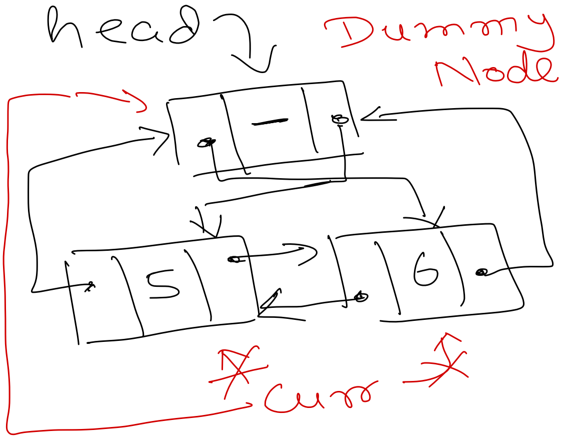
① Set current node's previous node's next to newNode

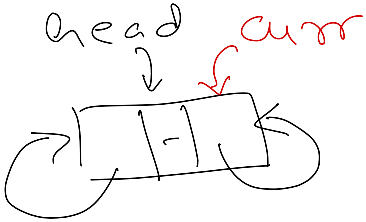
② Set newNode's previous to current node's previous; Node 10

③ Set newNode's next to curr.

④ Set current's previous to newNode.

Befor Step #4, Step # 1 should be done.





Empty circular list

```
Forward Traversal - Circular Doubly List
- Set current to head node's next.
- while current is not head node do
  - Process current node.
  - Set current to current node's next.
```

— circular doubly list with dummy node.

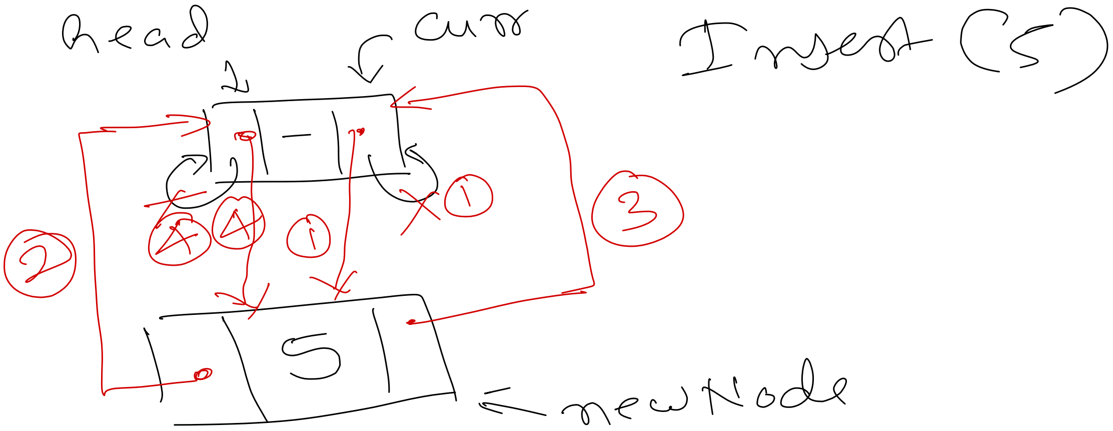
① Set current node's previous node's next to newNode

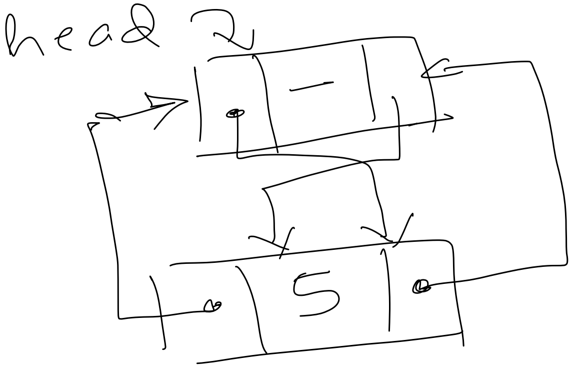
② Set newNode's previous to current node's previous. Node 10

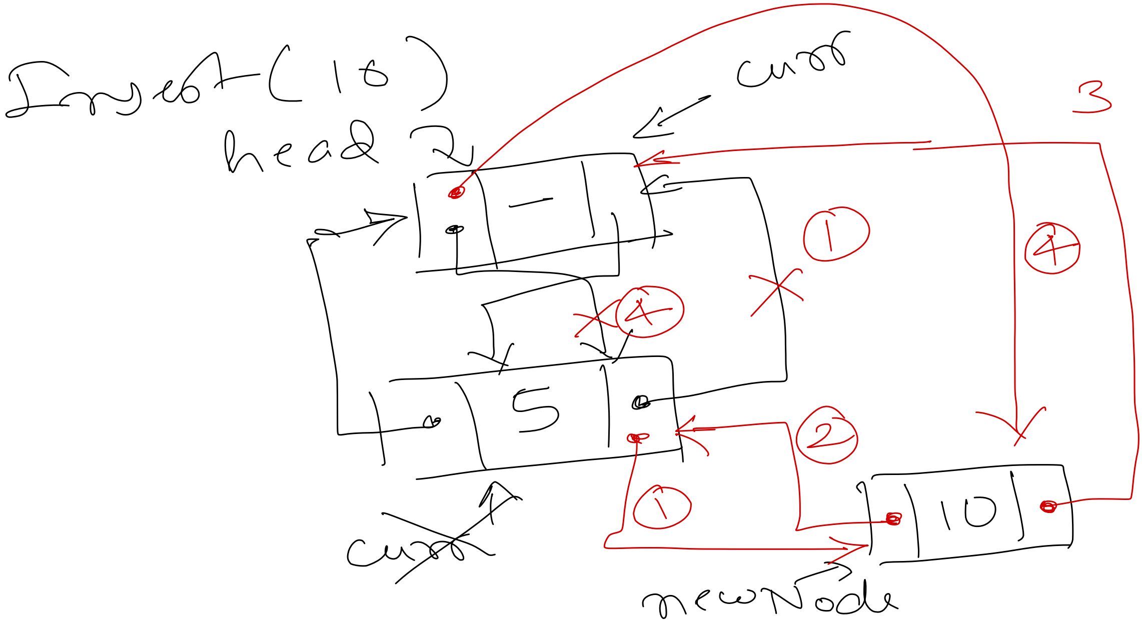
③ Set newNode's next to curr.

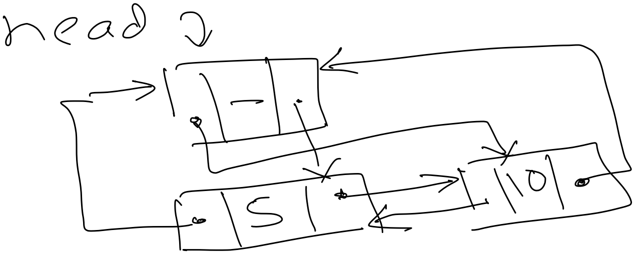
④ Set current's previous to newNode.

Before Step #4, Step # 1 should be done.


head
curr
Insert (5)
1
2
3
4
4
1
5
newNode


head
5

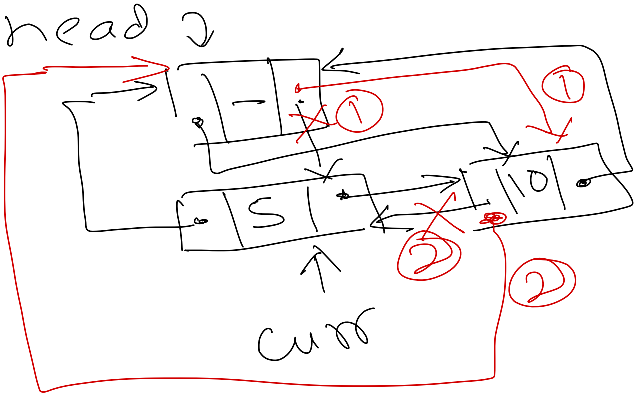

Insert (10)
curr
head
3
1
4
4
5
2
1
10
curr
newNode

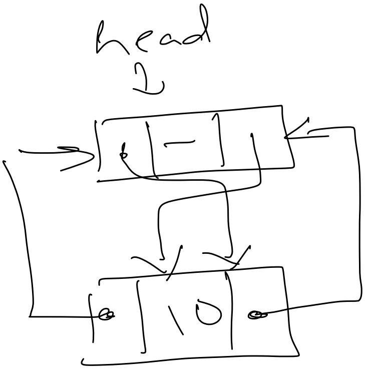


① Set $\overbrace{\text{current node's previous}}^{\text{head}}$ node's next to $\underbrace{\text{current node's next}}_{\text{Node 10}}$

② Set $\overbrace{\text{current node's next}}^{\text{Node 10}}$ node's previous to $\underbrace{\text{current node's previous}}_{\text{head}}$

Delete(5)





Reverse Traversal - Circular Doubly List - with dummy node.

```
- Set current to head node’s previous.
- while current is not head node do
  - Process current node.
  - Set current to current node’s previous.
```

Insert(element) - In circular doubly list with dummy node.

```
- Make space to store element, say newNode.
- Store element in newNode.
- Set current to first data node (first data node => head’s next).
- while current is not head do
    - if current node’s data > element then
        - End the loop.
    - Set current to current’s next node.
- Set (current node’s previous) node’s next to newNode.
- Set neNode’s previous to current node’s previous.
- Set newNode’s next to current.
- Set current node’s previous to newNode.
```

Delete(element) - In circular doubly list with dummy node.

```
- Set current to first data node (first data node => head’s next).
- while current is not head do
    - if current node’s data = element then
        - End the loop.
    - Set current to current’s next node.
- if current node is head then
    - Stop. // Element not found.
- Set (current node’s previous) node’s next to (current node’s next).
- Set (current node’s next) node’s previous to (current node’s previous).
```