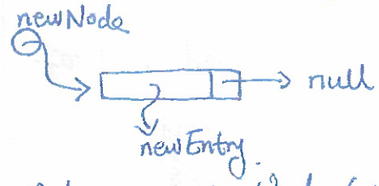
Insert a new node into list L

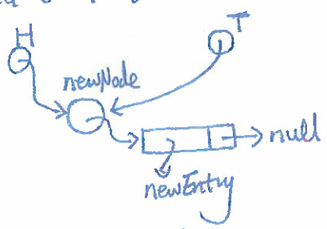
Step1: Create a new node with data newEntry.




```
Node newNode = new Node(newEntry);
```

L is empty

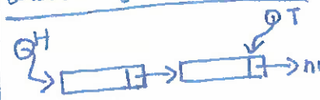
Step2: Connect both head and tail to the newNode



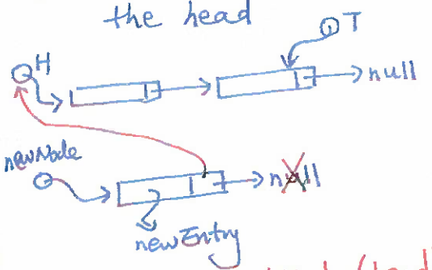

```
head = newNode
tail = newNode
```

L is not empty

Insert front of L

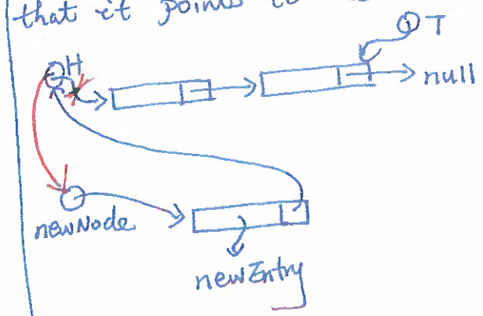


Step2: Connect newNode with the head



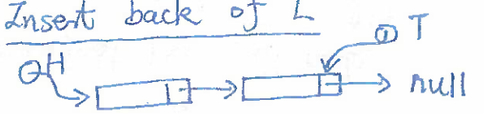

```
newNode.setNextNode(head)
```

Step3: Readjust the head so that it points to the newNode



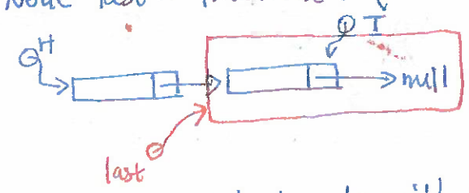

```
head = newNode
```

Insert back of L

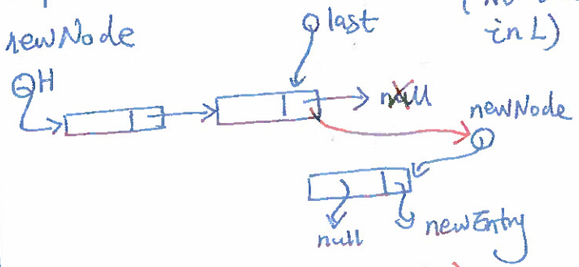


Step2: find the last node in the list (No tail node in L)

```
Node last = findNodeAt(numEntries).
```

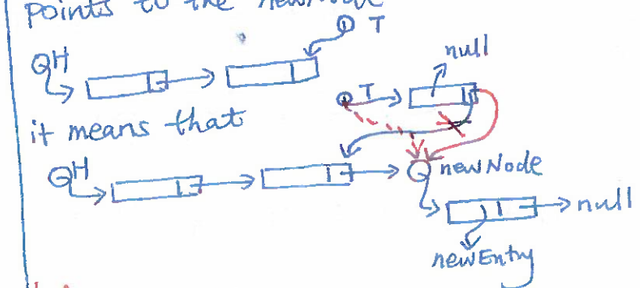


Step3: Connect last node with the newNode (No tail node in L)



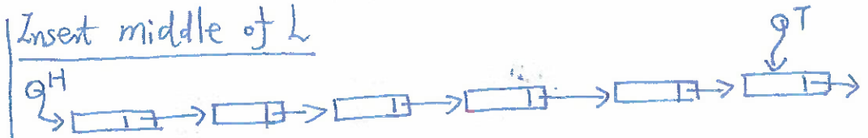

```
last.setNextNode(newNode)
```

Step4: If there is a tail node in L, we skip steps3 and 4; we directly readjust the tail node so that it points to the newNode

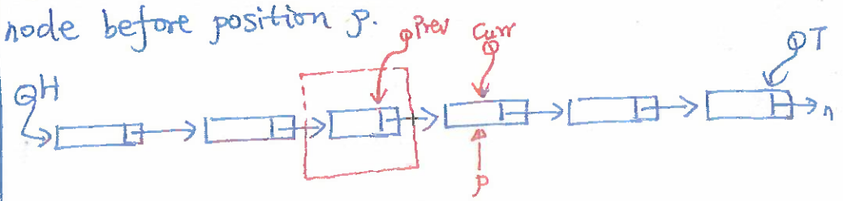



```
tail.setNextNode(newNode)
tail = newNode
```

Insert middle of L

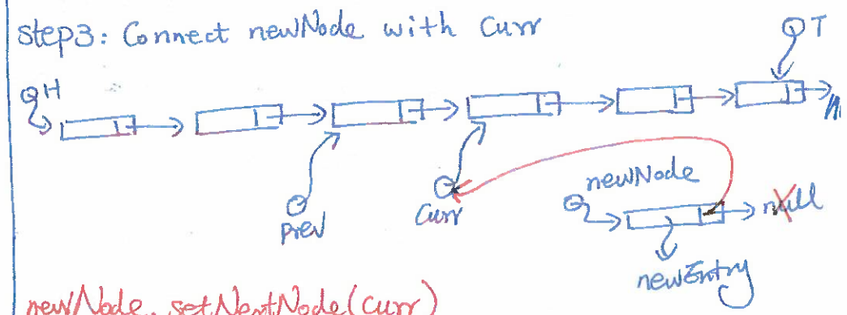


Step2: find the node in the current position p and th node before position p.



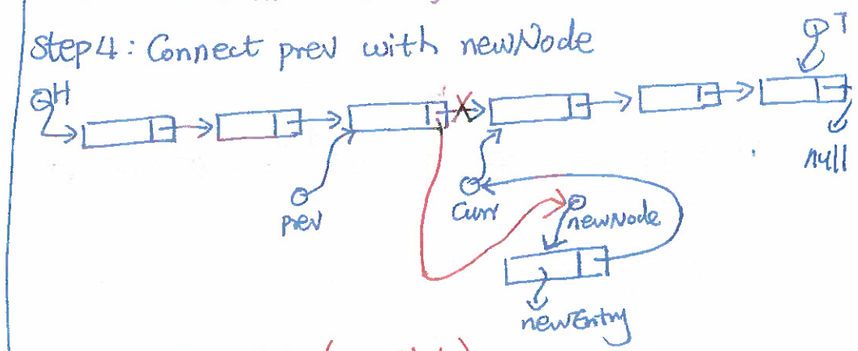

```
Node curr = findNodeAt(p)
Node prev = findNodeAt(p-1)
```

Step3: Connect newNode with Curr




```
newNode.setNextNode(curr)
```

Step4: Connect prev with newNode




```
prev.setNextNode(newNode)
```

Remove a node from List L

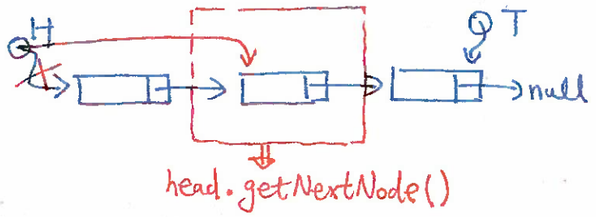
Precondition: L is not empty

Given the position of the removal node p.

P is the first node in L (i.e. p = 1)

L contains more than 1 node.

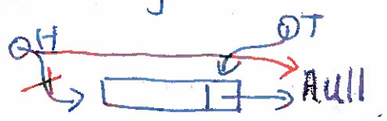
Step 1: Readjust the head node



head = head.getNextNode()

L Contains only 1 node

tep1: Re adjust the head node

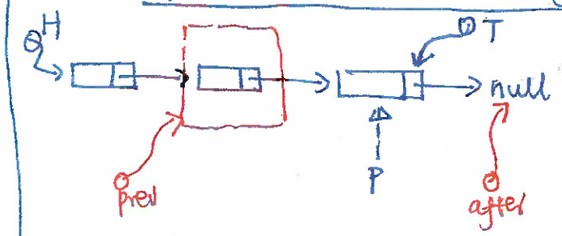


head = head.getNextNode()

Step2: Readjust the tail node

tail = Null

P is the last node of L



Step1: find the node before position p

Node prev = findNodeAt (p-1)

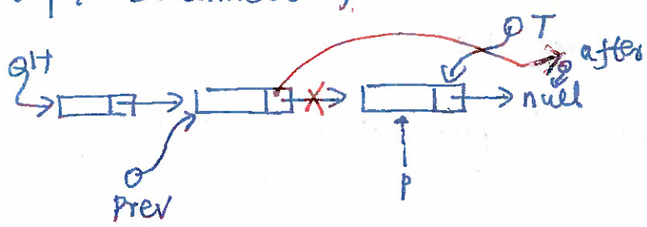
Step 2: find the node after position p

Node after = findNodeAt (p+1)

Step 3: Retrieve the node at position p.

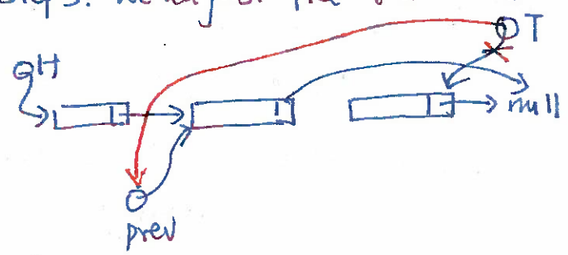
Node curr = prev.getNextNode().

Step4: Disconnect prev and curr



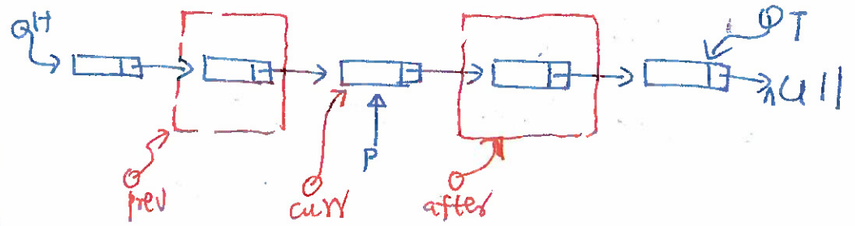
prev.setNextNode(after)

Step 5: Readjust the tail node



tail = prev

P is in the middle node of L



Step1: find the node before position p

Node prev = findNodeAt (p-1)

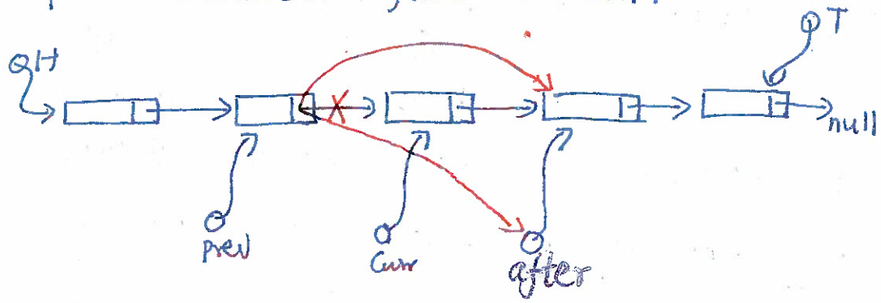
Step2: find the node after position p

Node after = findNodeAt (p+1)

Step3: Retrieve the node at position p

Node curr = prev.getNextNode()

Step4: Disconnect prev and curr.

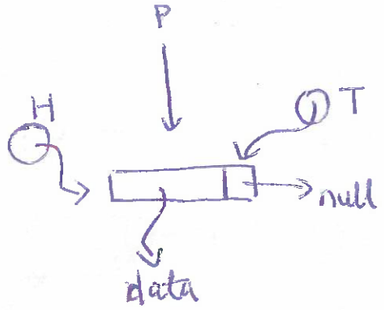


prev.setNextNode(after)

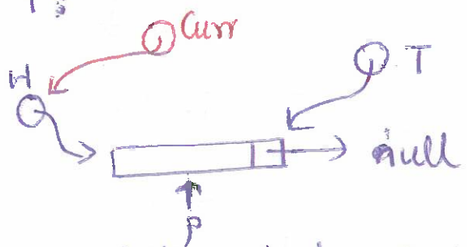
Traverse a whole list L

Precondition: the list L is not empty.

Given the destinational position p where we stop the traversing.



If L contains only one node and P is 1:



Step 1: Need a tracker curr to keep track the traversing and curr refers to the head.

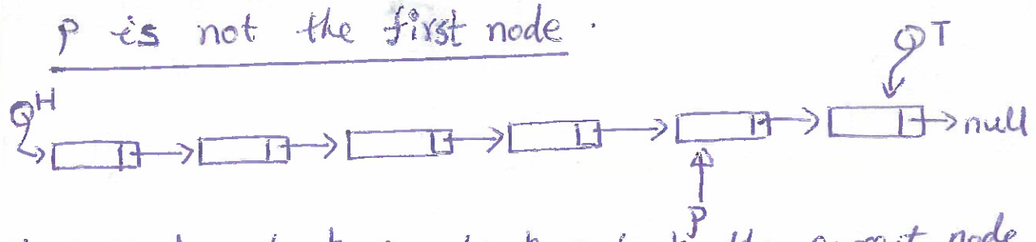
```
Node curr = head
```

Step 2: As L contains only 1 node, Curr is the node we want to return.

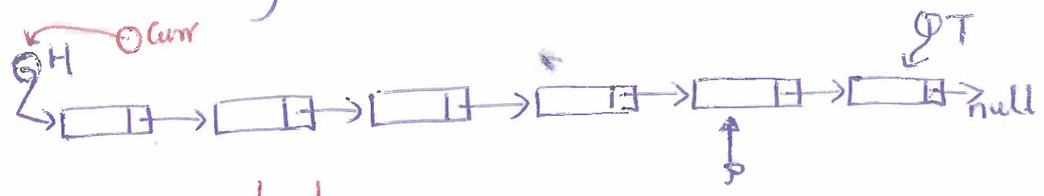
```
return Curr
```

L contains more than 1 node

p is not the first node.

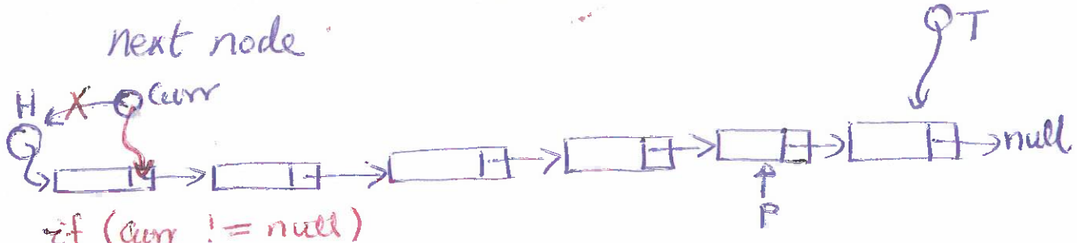


Step1: Need a track curr to keep track the current node we are traversing and Curr starts with the head node



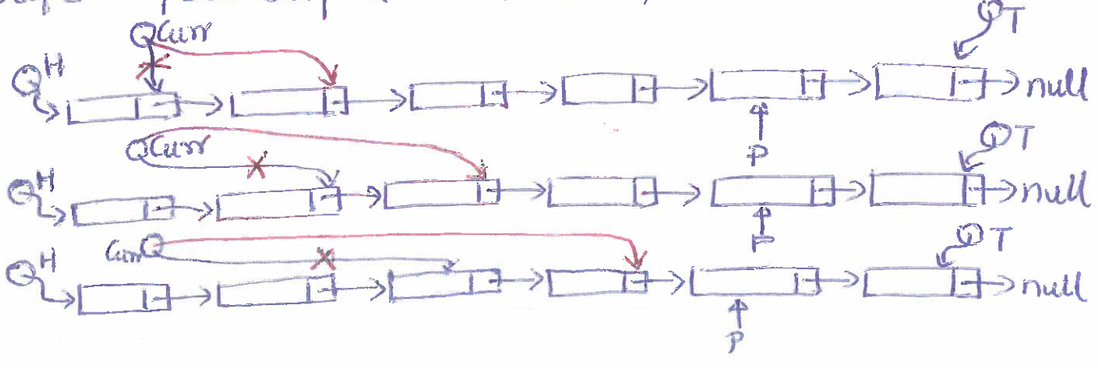

```
Node Curr = head
```

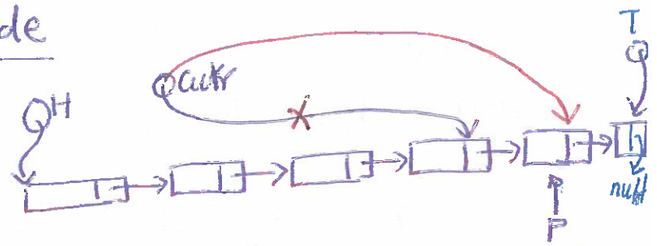
Step2: Check if curr node is null or not, if it is not null, update curr to the next node




```
if (Curr != null)
    Curr = Curr.getNextNode()
```

Step3: Repeat step 2 until the position reaches p-1






```
for (int pos=1; pos < P; pos++)
    if (Curr != null)
        Curr = Curr.getNextNode()
```

How to check if list L contains an element elem?

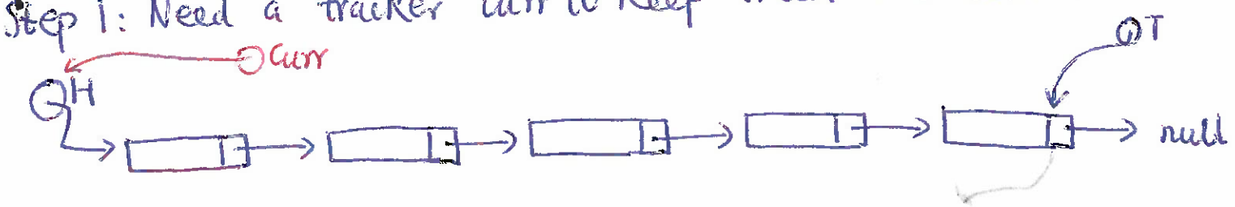
Given an element elem to be searched for

L is empty

Return false

L is not empty

Step 1: Need a tracker curr to keep track the current node we are visiting.

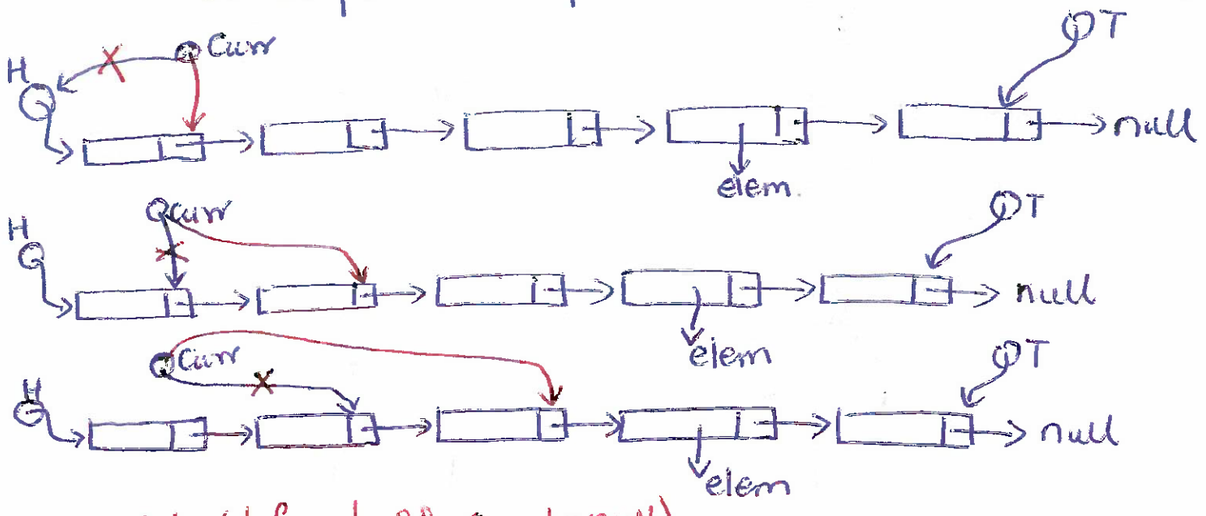


Node curr = head

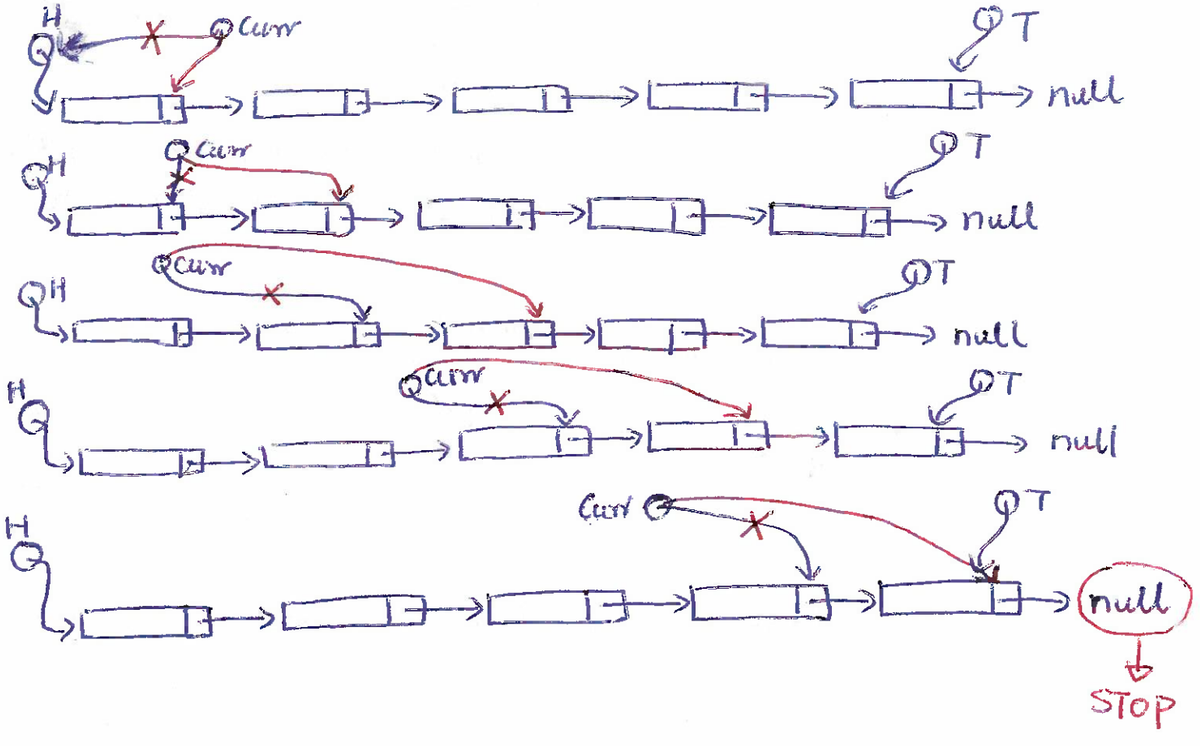
Step 2: Create a boolean flag to indicate whether the elem is found or not.

boolean found = false

Step 3: Repeatedly go through each node in L and check if it contains elem or not
it stops the repetition either elem is found or it reaches the last node in L



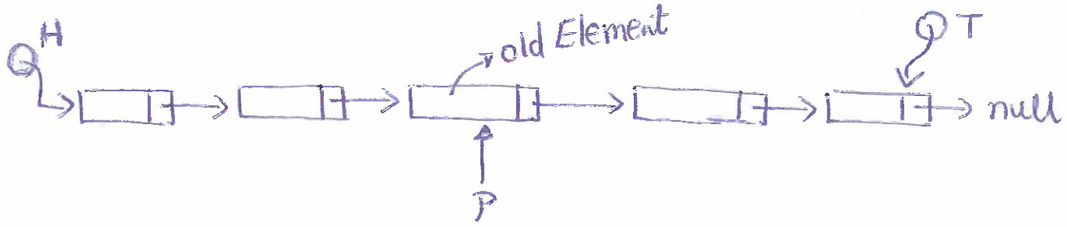

```
while (!found && curr != null)
    if (elem.equals(curr.getData()))
        found = true
    else
        curr = curr.getNextNode()
```



Replace an element with a new one in List L

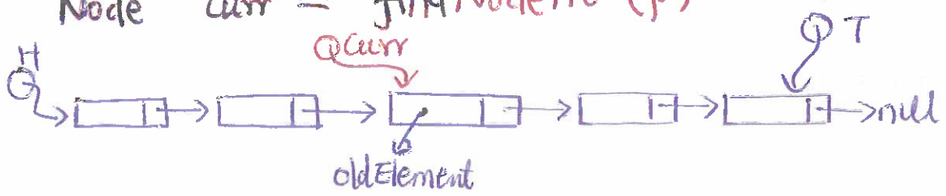
Precondition: L is not empty.

Given the position of the element to be replaced p.

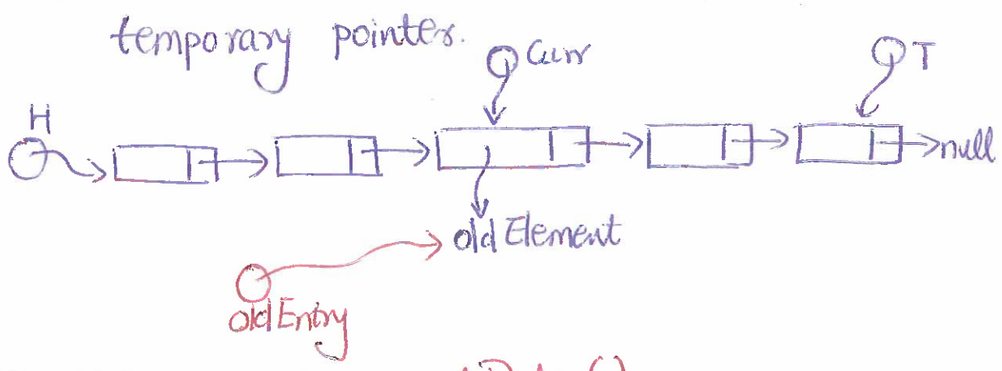


Step 1: Find the node at position p

Node curr = findNodeAt (p)

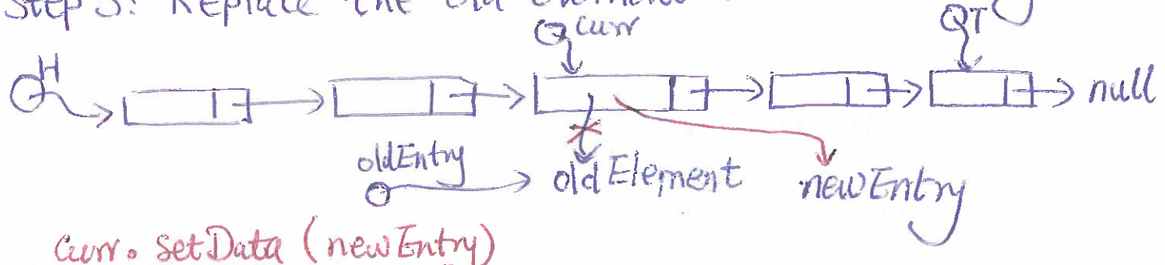


Step 2: Store the data of curr node into a temporary pointers.



T oldEntry = curr . getData ()

Step 3: Replace the old Element with new Entry



curr . setData (newEntry)

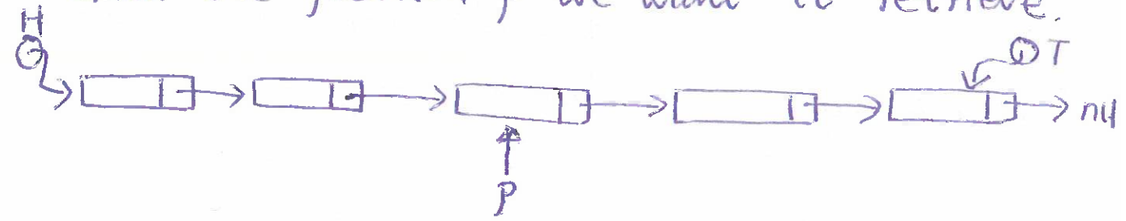
Step 4: Return the original data before the replacement

return oldEntry

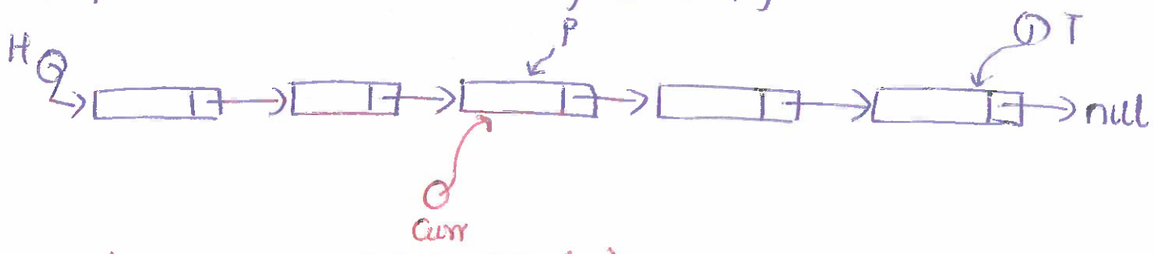
Retrieve the data at Specific position L.

Precondition: L is not empty

Given the position p we want to retrieve.

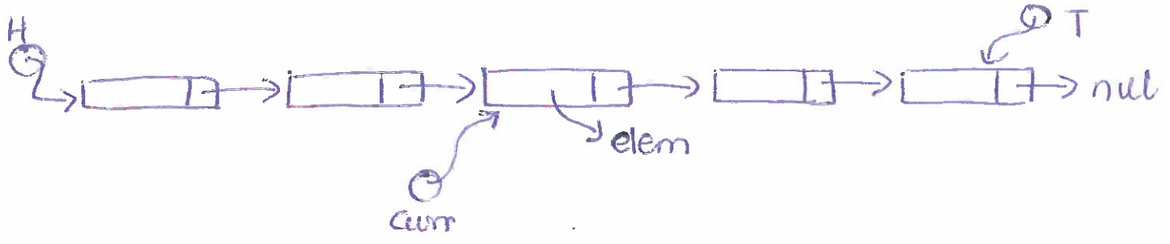


Step 1: Find the node at position p



Node curr = findNode At (p)

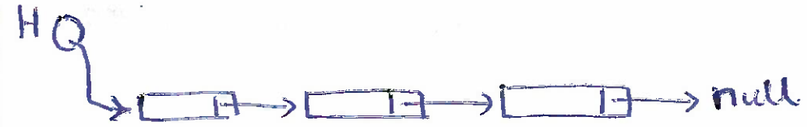
Step 2: Return the data Contained in node curr



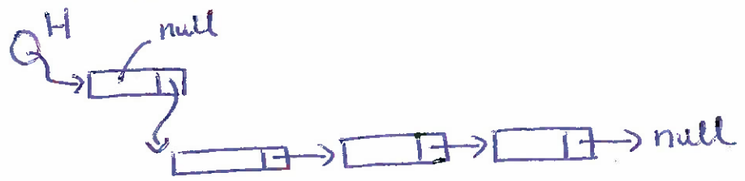
return curr . getData ()

How to remove all nodes in list L?

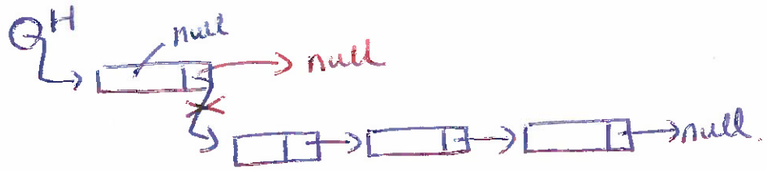
Do nothing !!



It actually means that



Step 1: disconnect head with the whole list.

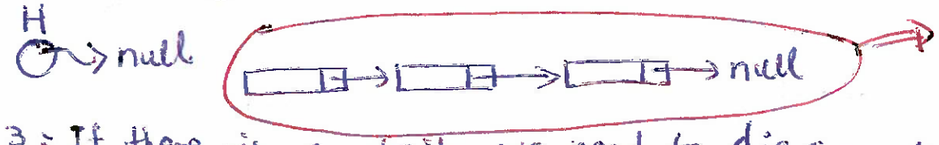


head.setNextNode(null)

|||

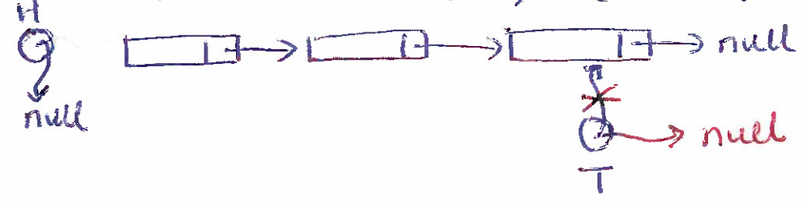
head = null (as both data and next pointer are null we can simplify it to head = null, no need to do head.setNextNode(null))

Step 2: Garbage Collector will claim back the disjointed list L. intemally !!



Garbage Collector will automatically claim the memory back and free up memory.

Step 3: If there is a tail, we need to disconnect the tail with List L.



tail = null. (Same principle as head, no need to call tail.setNextNode(null))!!!