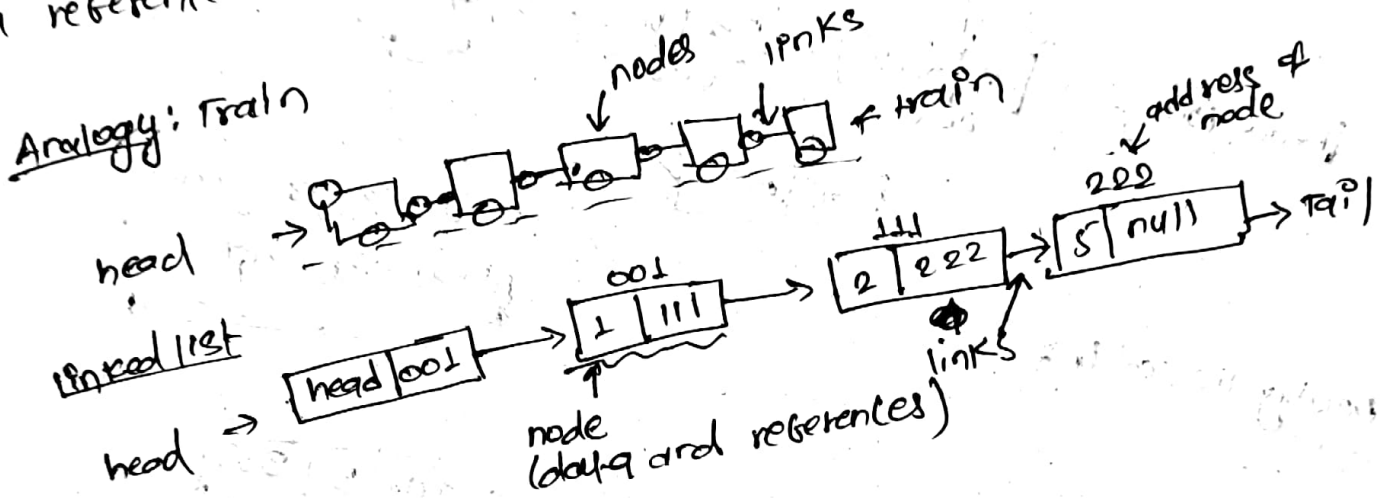
## 2. Linked list

linked list is a form of sequential collection and it does not have to be in order. It is made up of independent nodes that may contain any type of data and each node has a reference to the next node in the link.



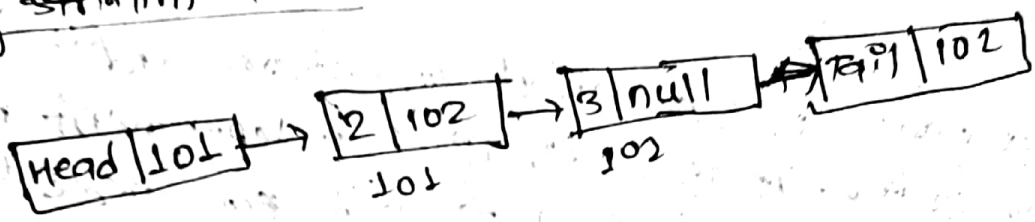

## linked list vs. Array

- Elements of linked list are independent objects
- variable size : the size of linked list is not predefined.
- random access : accessing an element is very efficient in arrays.

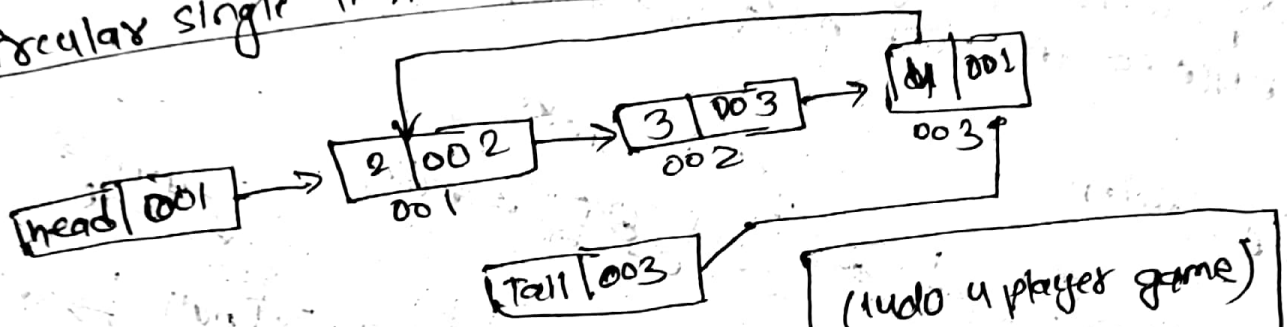
## Types of linked list

1. singly linked list
2. circular singly linked list
3. Doubly linked list
4. circular Doubly linked list

1. Singly linked list



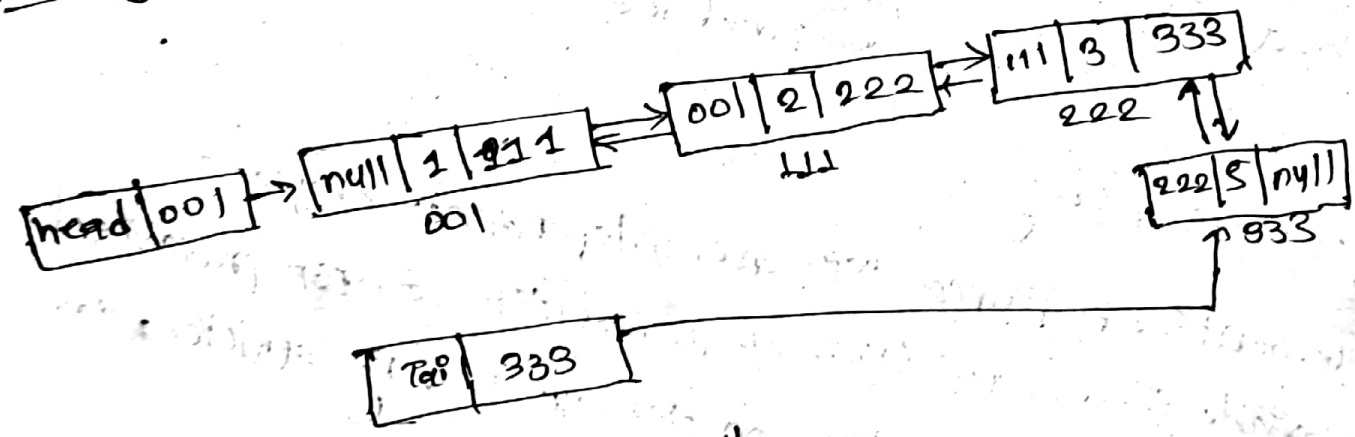

2. circular single linked list



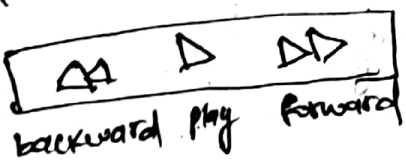

(ludo 4 player game)

P1 → P2 → P3 → P4

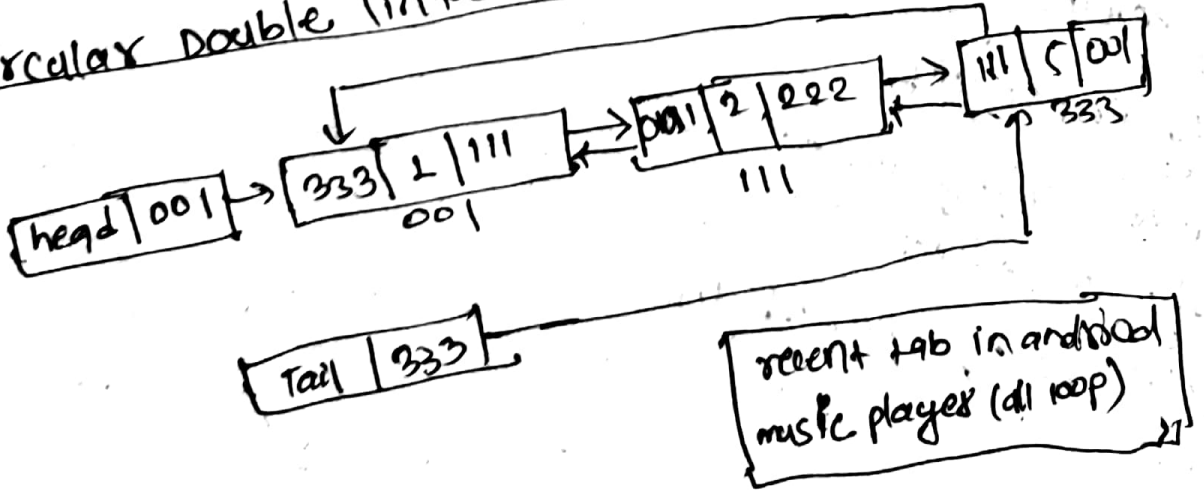
3. Doubly linked list




- Traversing back and forth
- music player

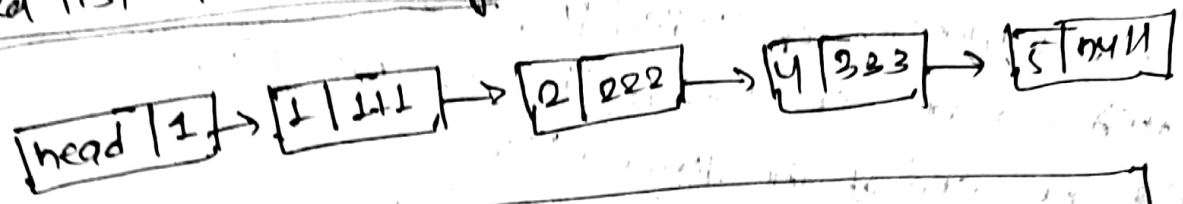


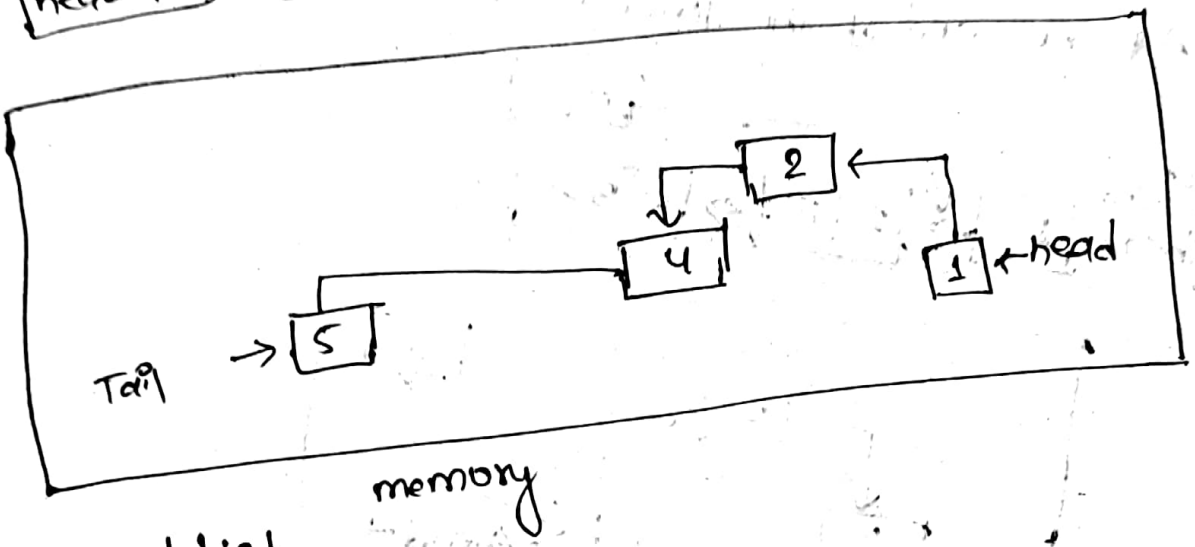

4. Circular Double linked list




recent tab in android music player (dl loop)

## Linked list in memory

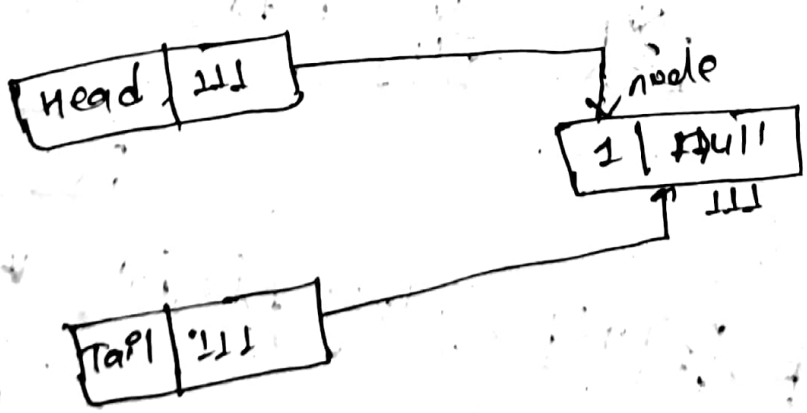



## 1. Singly Linked list

### 1. Creating single linked list

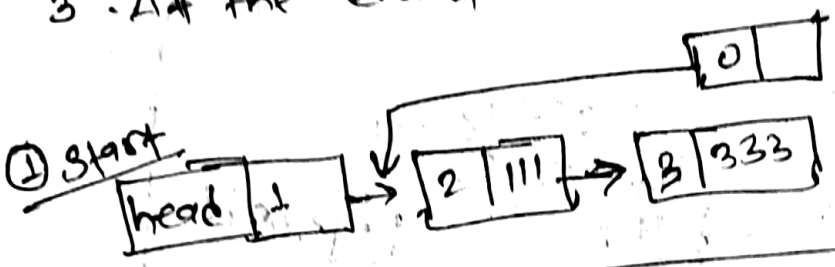
1. Create Head and Tail, initialize with null
2. Create a blank node and assign a value to it and reference to null.
3. Link Head and Tail with these node.



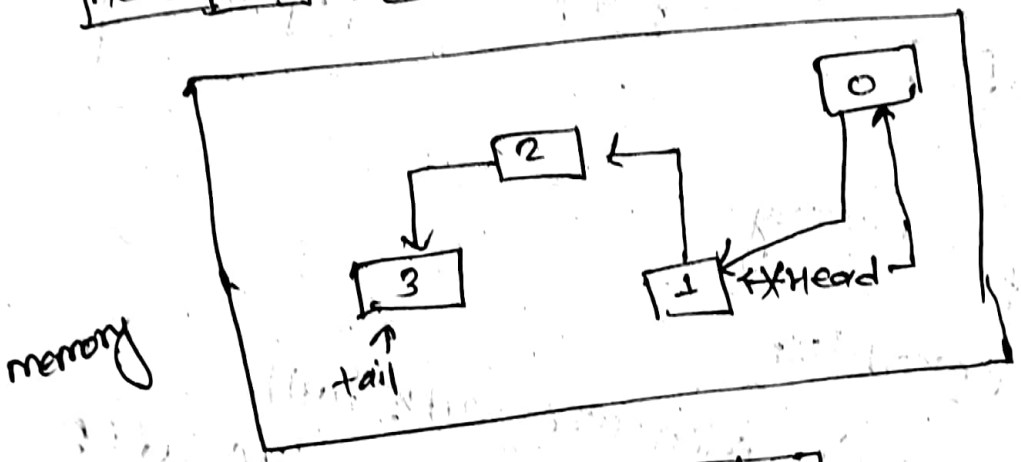

## code in pc

## 2. Insertion to Linked list in memory

1. At the beginning of linked list
2. After a node in the middle of linked list
3. At the end of the linked list

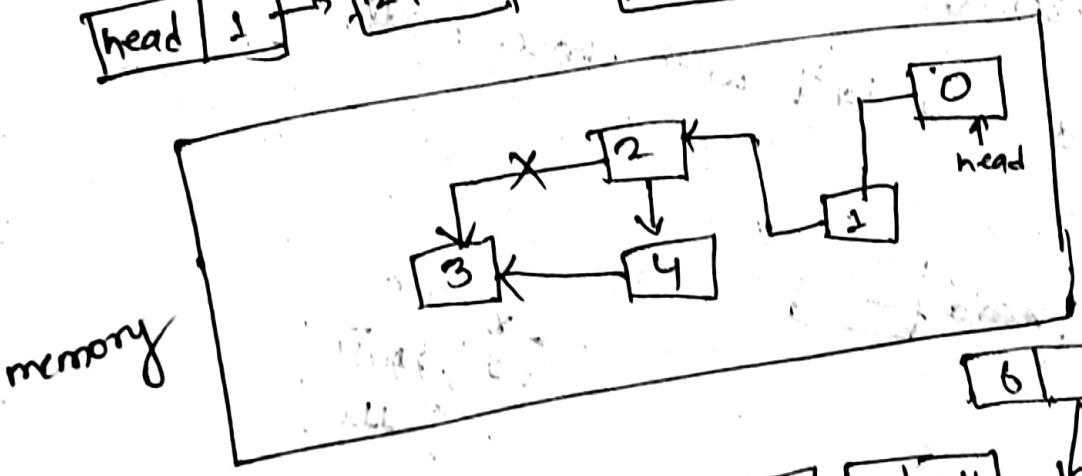



① start





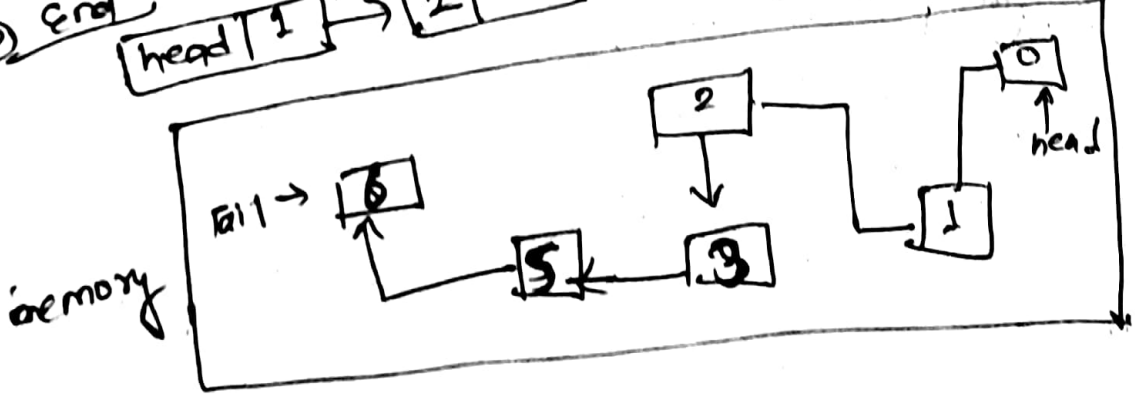
② middle



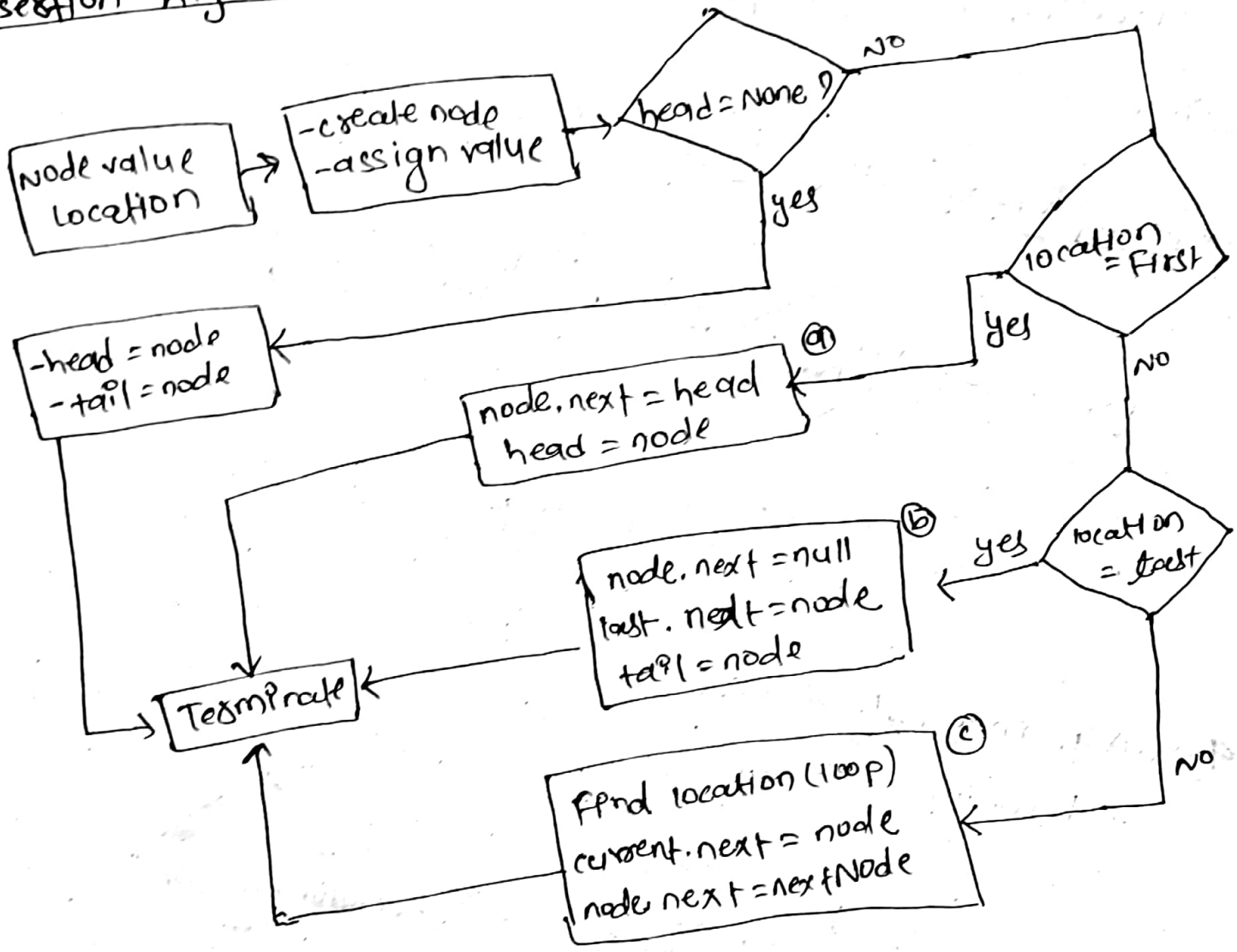


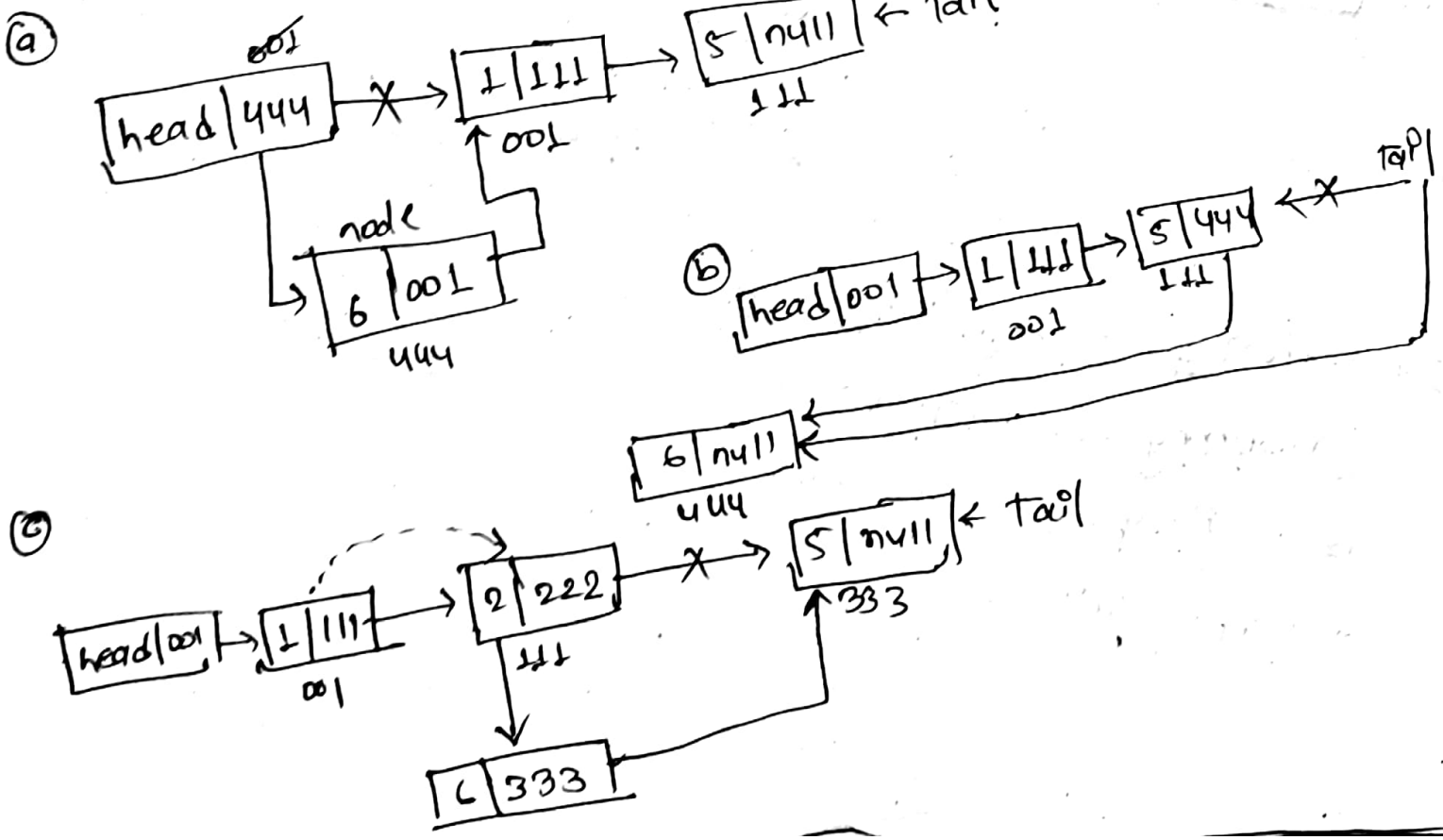


③ end



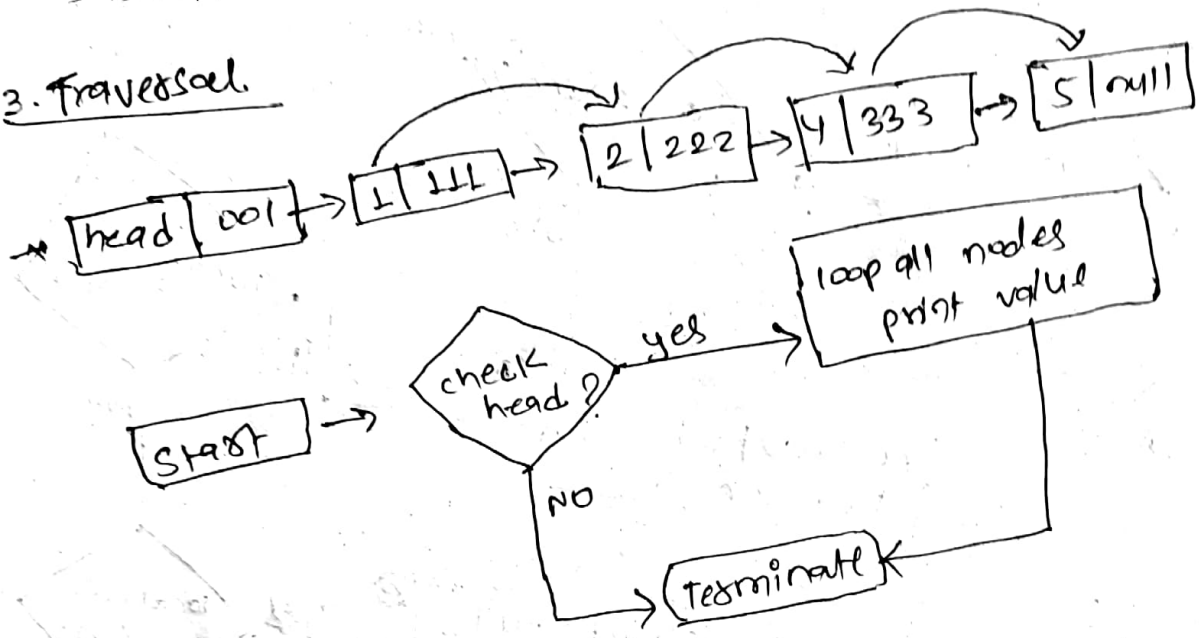
# Insertion Algorithm



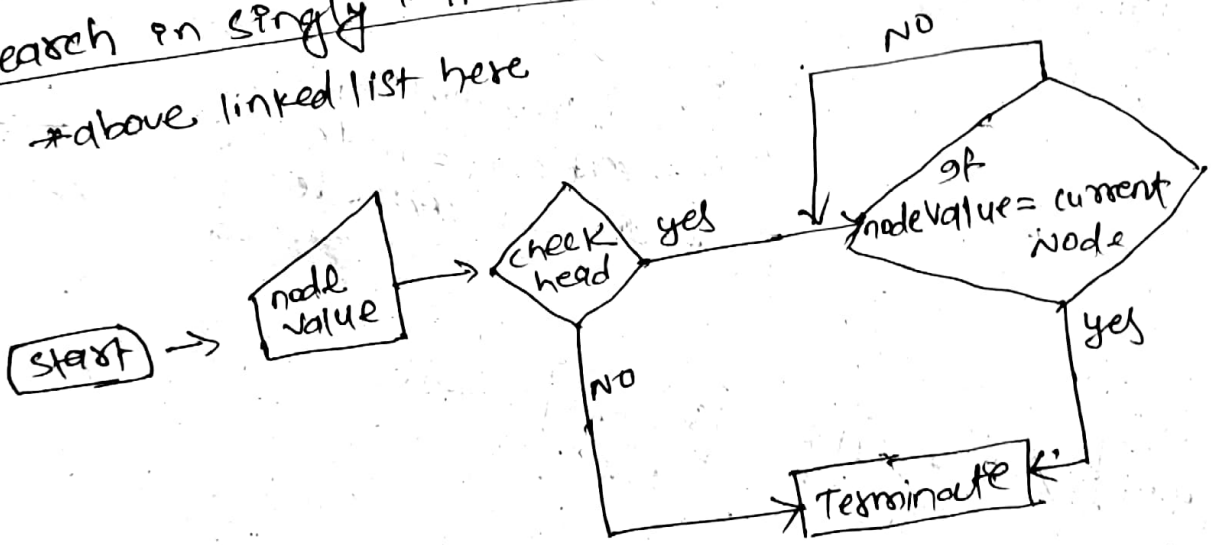
insertion method in pc.

3. Traversal




4. Search on singly linked list

*above linked list here




5. Deletion on singly linked list

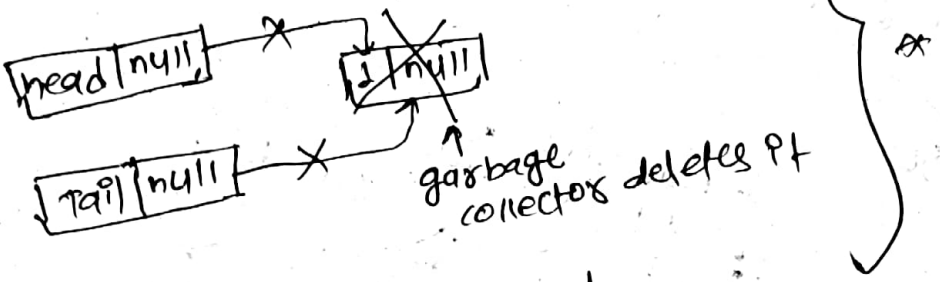
(a) Deleting the first node

(b) Deleting any given node

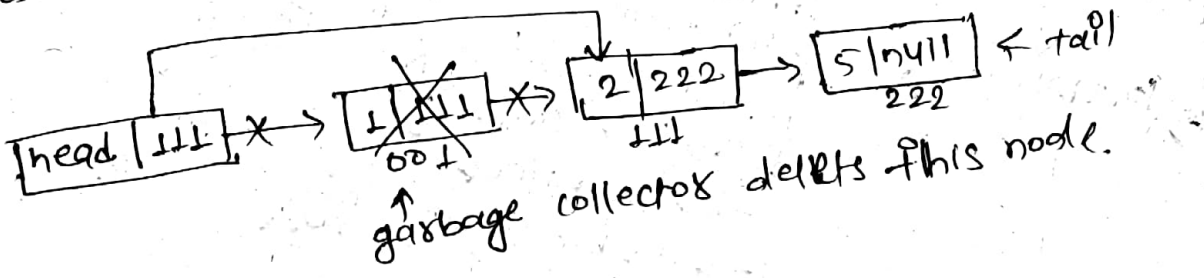
(c) Deleting last node.

④ Deleting 1st node

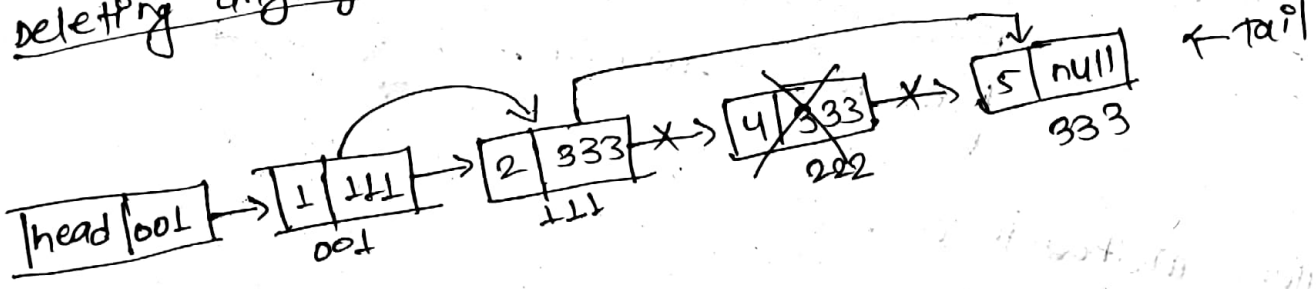
case 1: only one node




case 2: more than one node




⑤ Deleting any given node

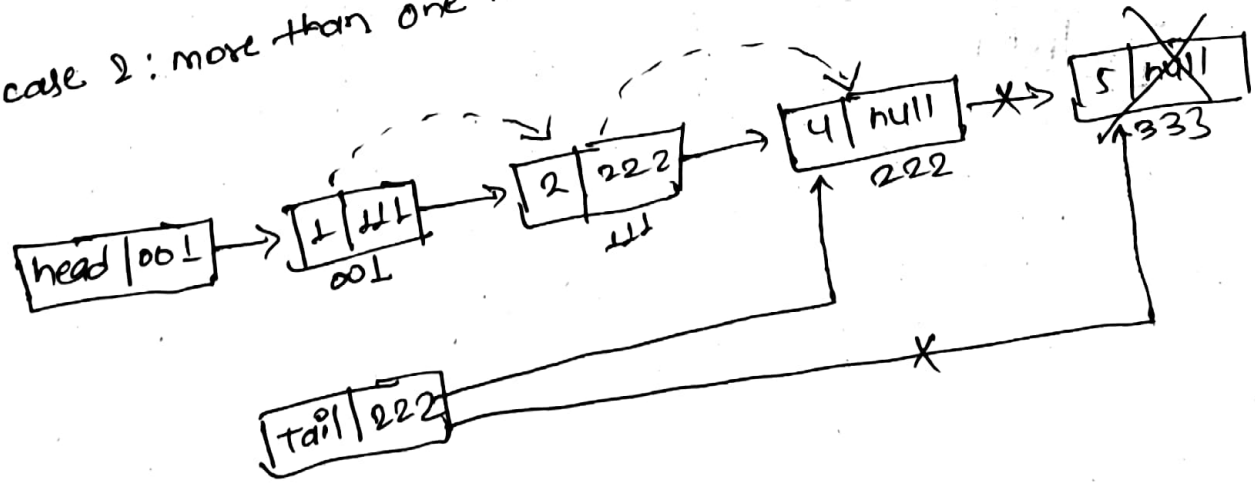



⑥ Deleting last node

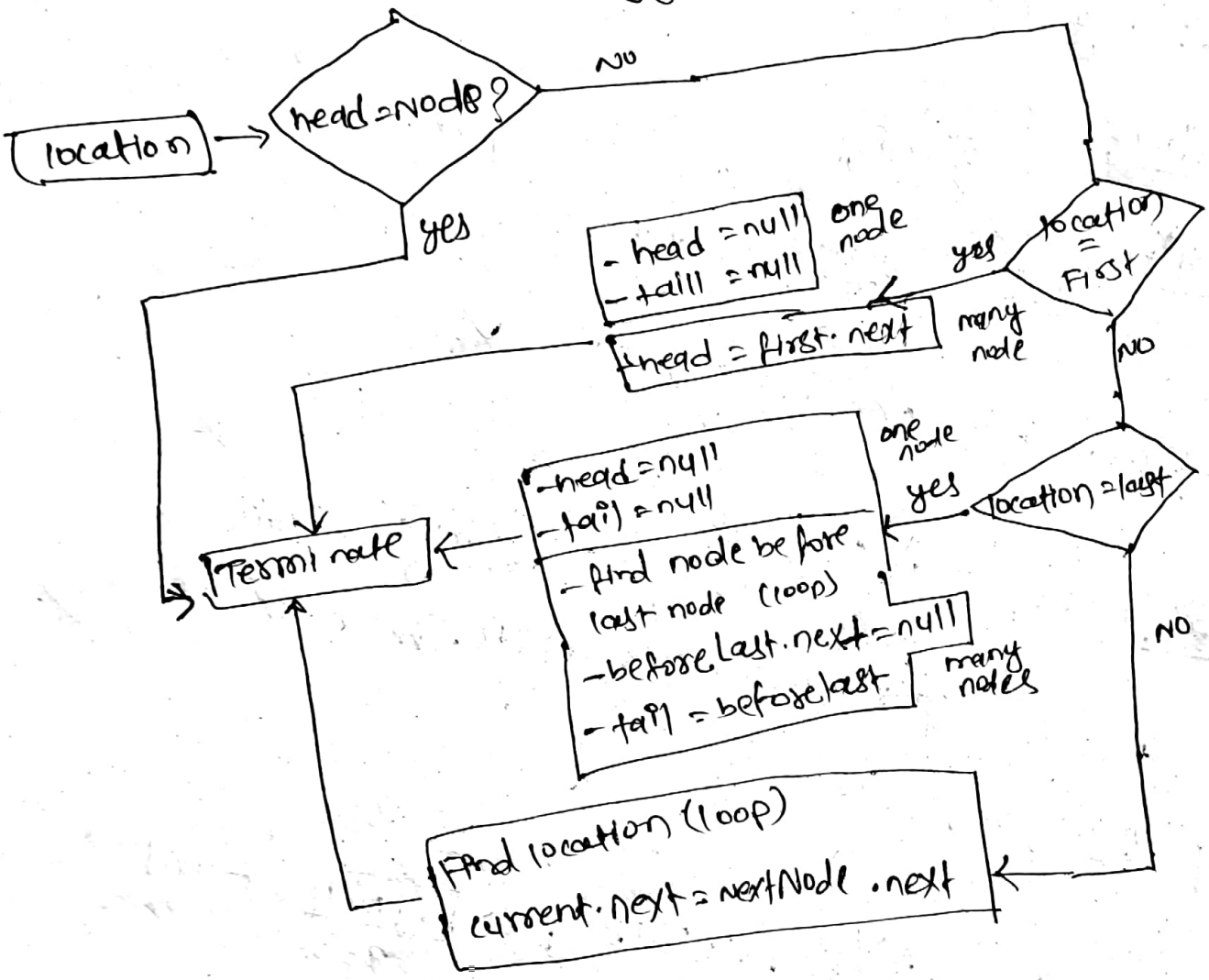
case 1: only one node

* same as above

case 2: more than one node



# Deletion Algorithm for singly linked list




Deletion method in PC.

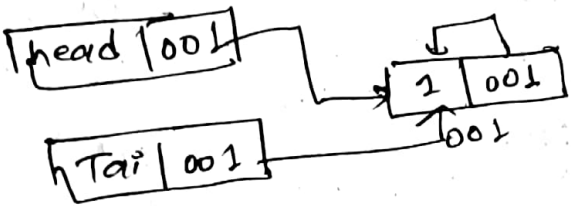
## 6. Deletion of entire single linked list

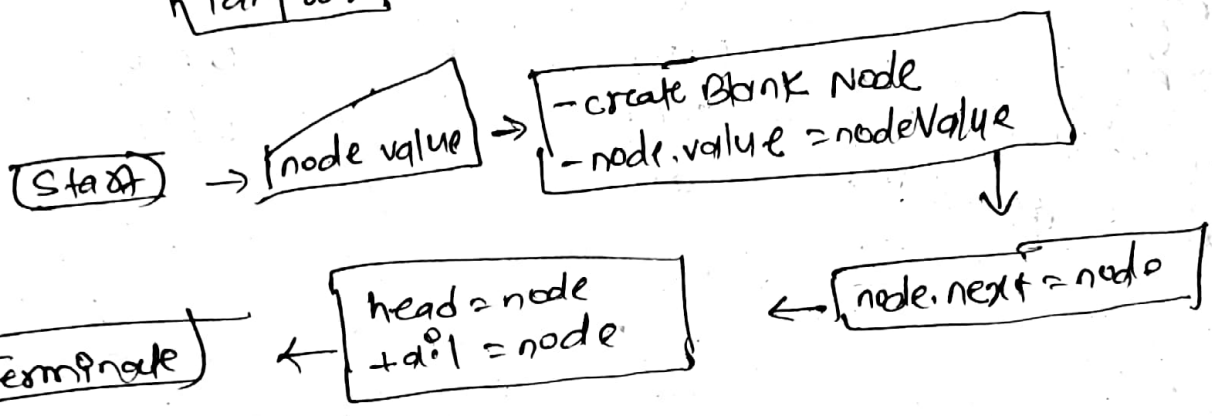
# Time and Space complexity of SLL

| singly linked list | Time complexity | space complexity |
|---|---|---|
| creation | O(1) | O(1) |
| Insertion | O(n) | O(1) |
| searching | O(n) | O(1) |
| Traversing | O(n) | O(1) |
| Deletion of a node | O(n) | O(1) |
| Deletion of linked list | O(1) | |

# 2. Circular linked list

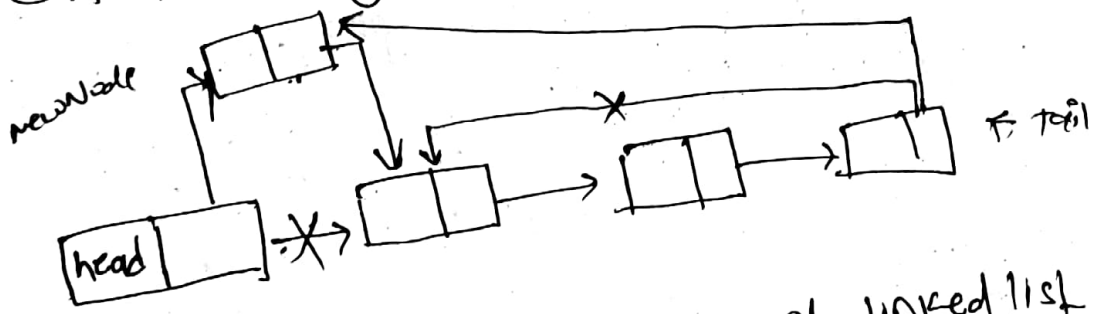
## 1. create CSLL



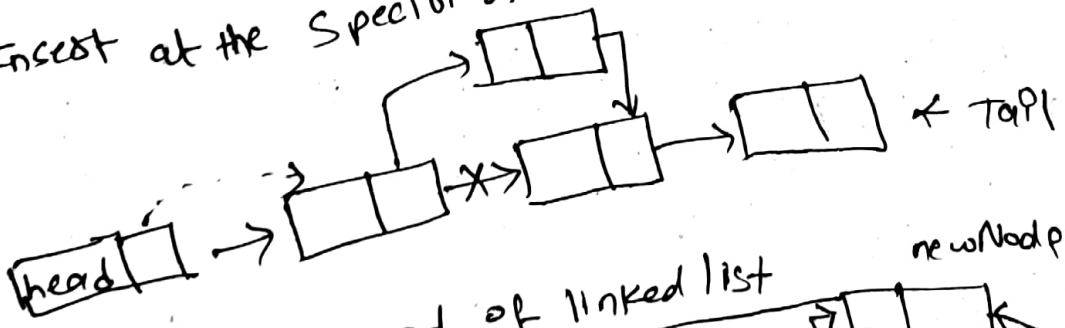




## 2. Insertion - CSLL

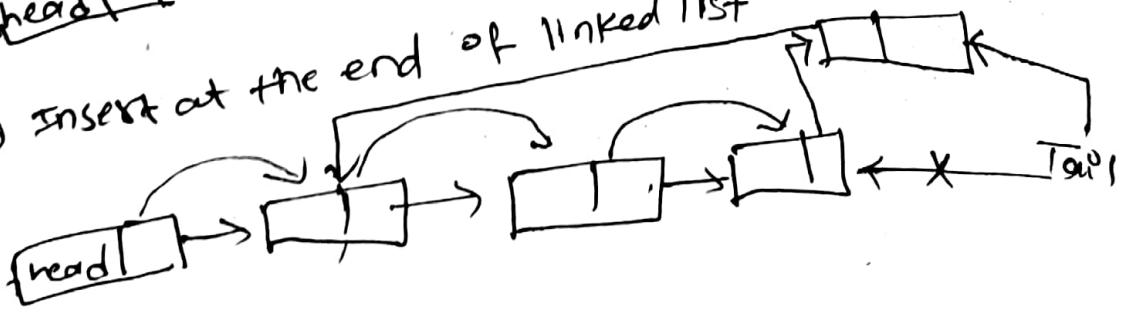
(a) At the beginning of linked list



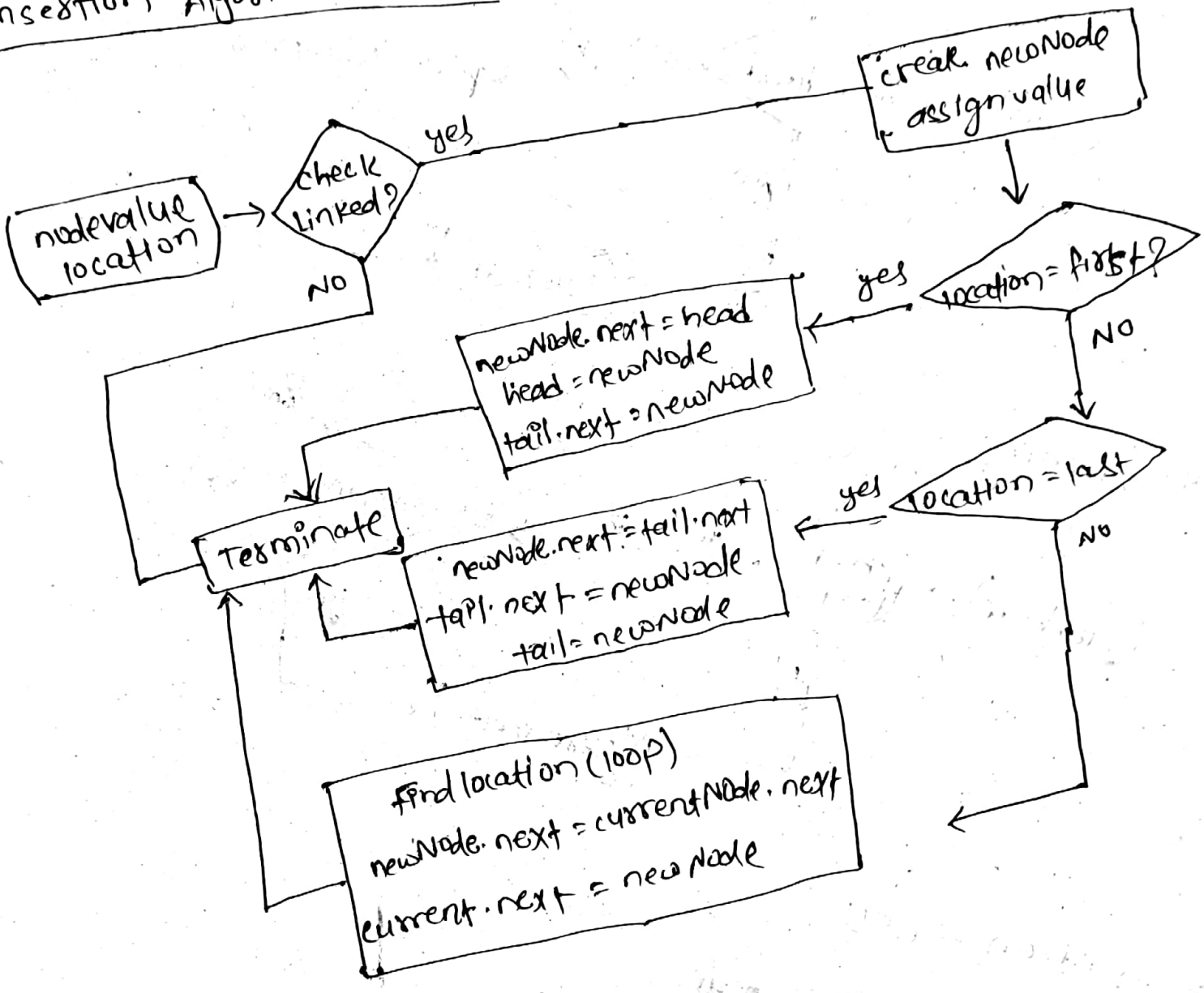

(b) Insert at the specified location of linked list




(c) Insert at the end of linked list



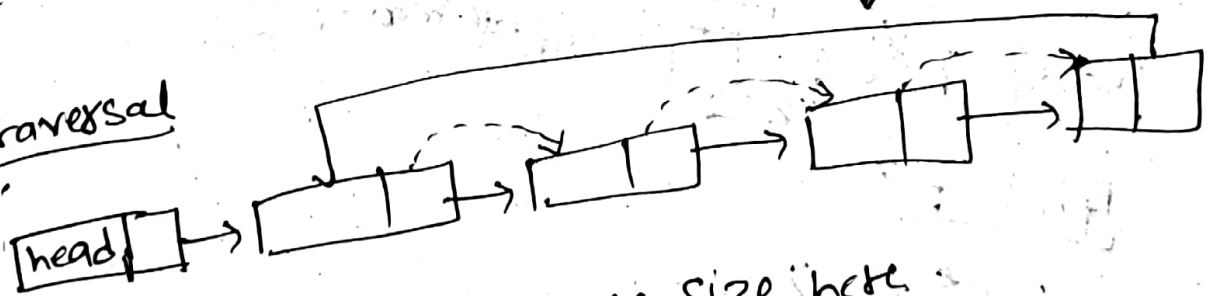
# Insertion Algorithm in CSLL



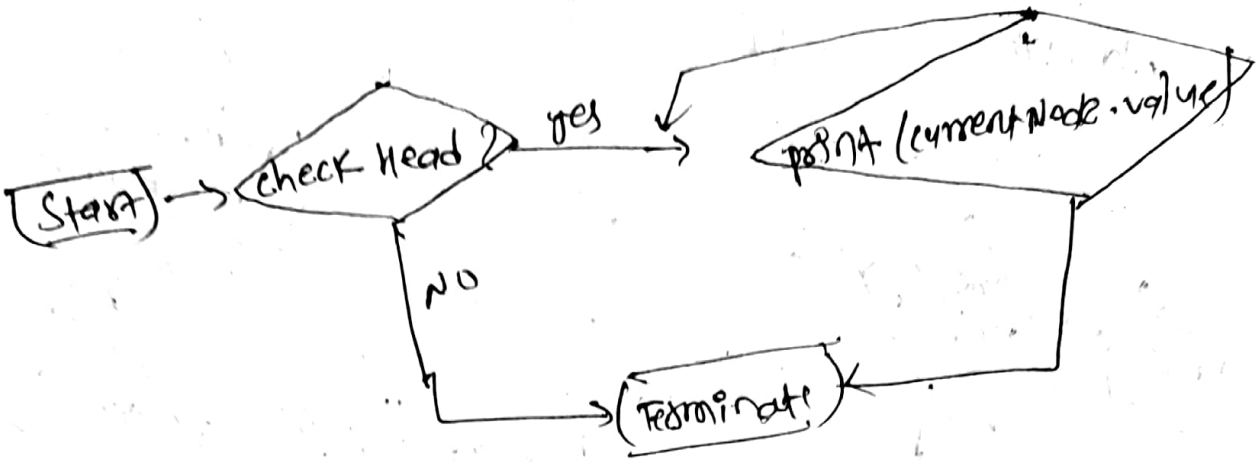

code in pc!

THIS cause infinite loop.

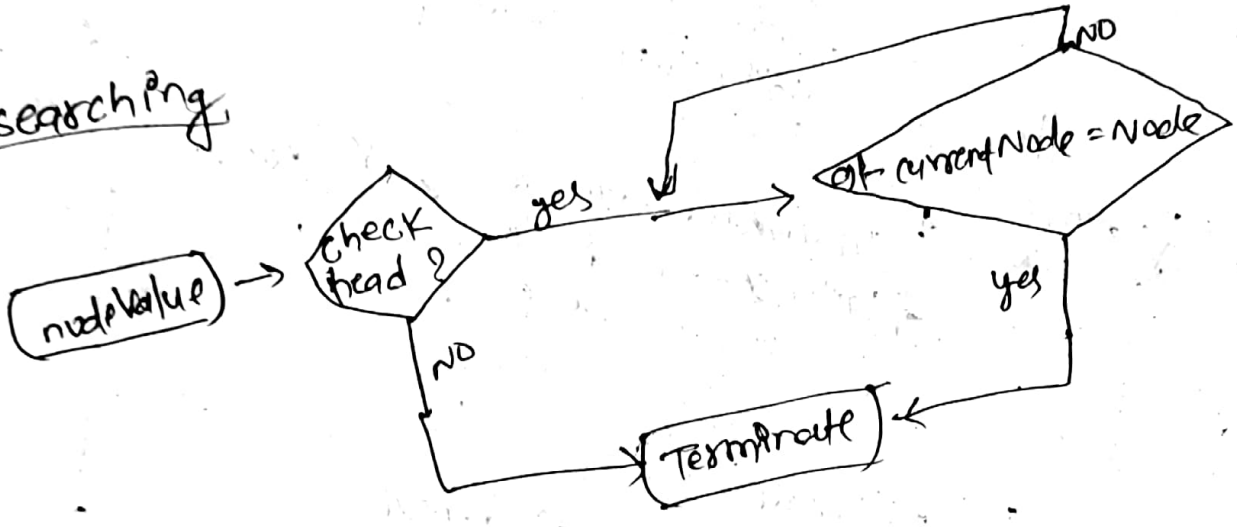
## 3. Traversal




To prevent infinite loop, we use size here.



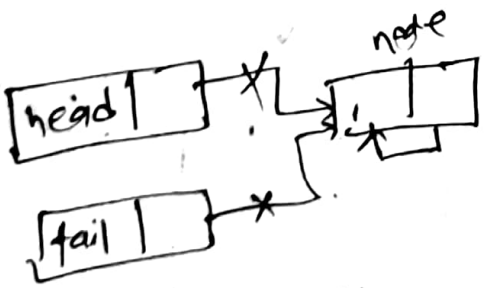

4. searching




5. Deletion - CSLL

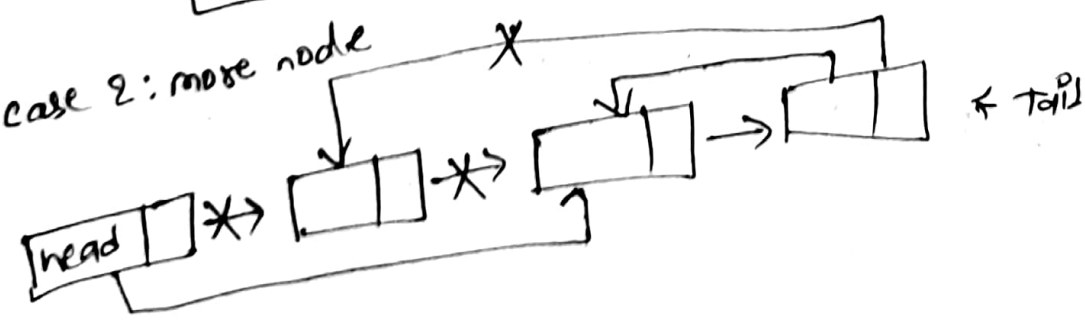
(a) Deleting the first node.

case 1 : one ~~one~~ node

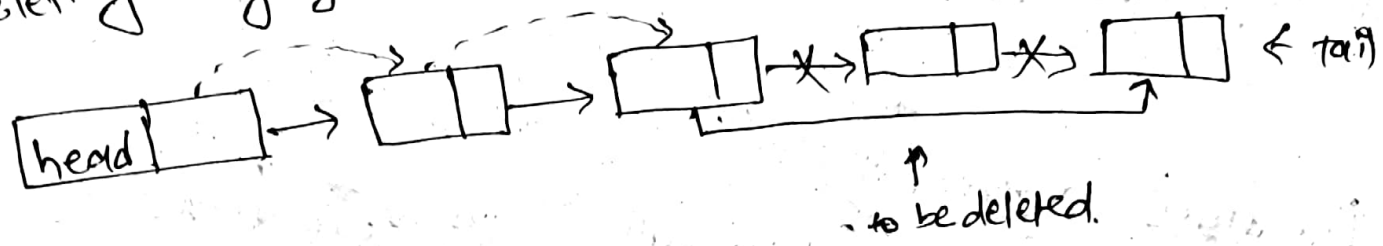



update head & tail to null.

case 2 : more node



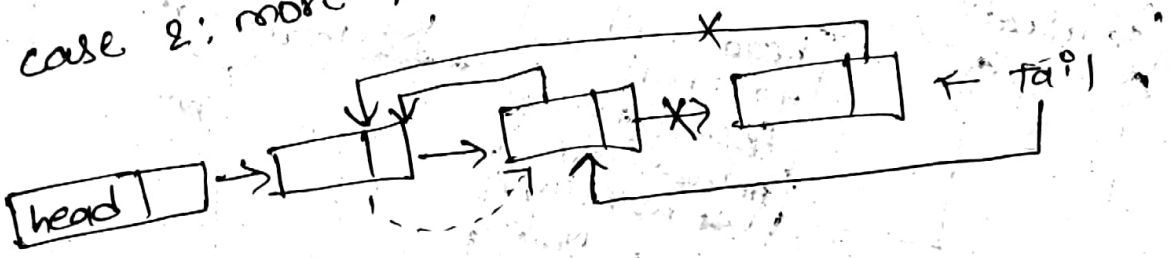
⑥ Deleting any given node




⑦ Deleting last node

case 1 : one node (same as ⑥)

case 2: more node



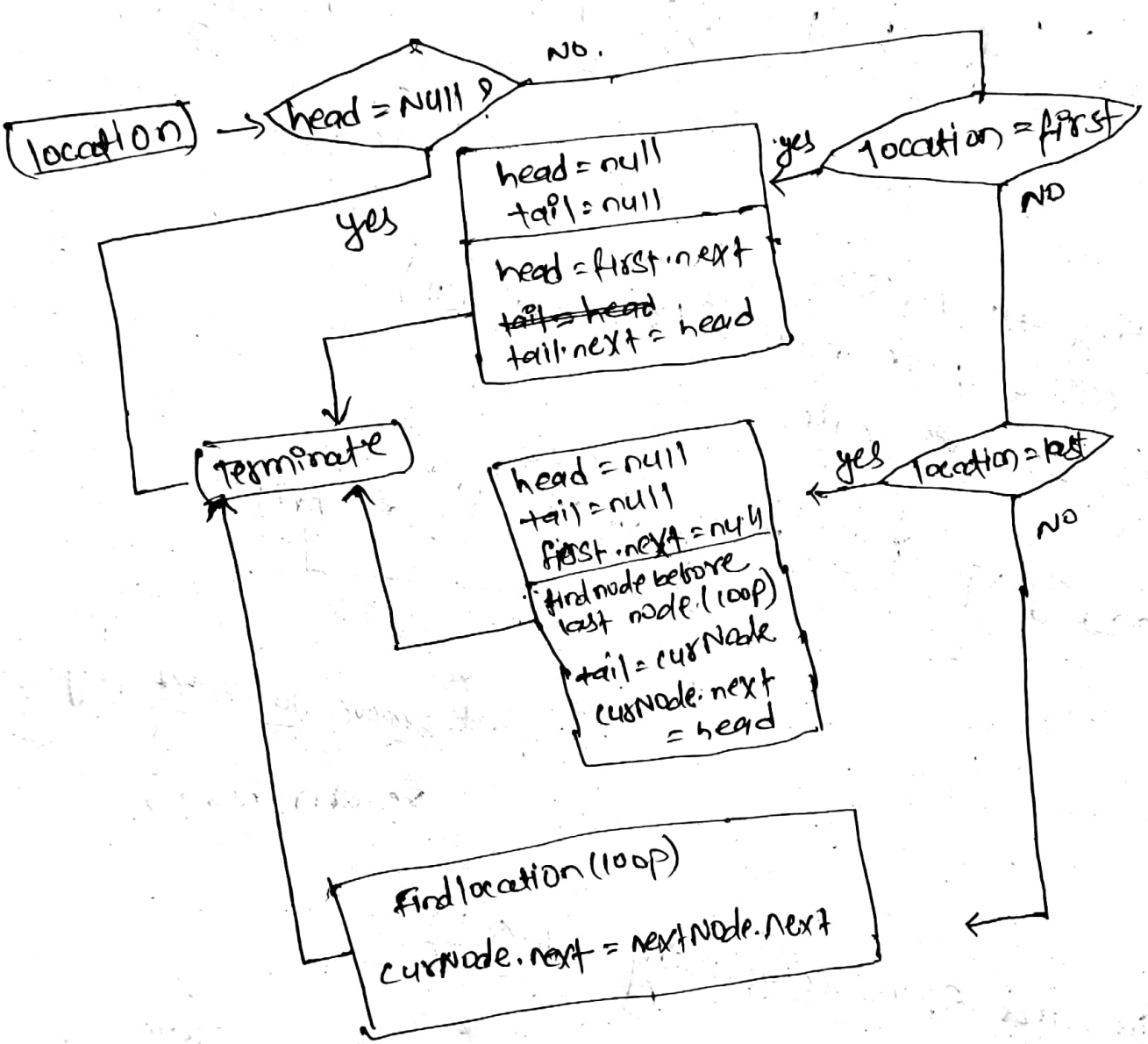

↳ move to next page

return here ;

## Time and space complexity of CSLL

| CSLL | Time complexity | space complexity |
|---|---|---|
| creation | O(1) | O(1) |
| Insertion | O(n) | O(1) |
| searching | O(n) | O(1) |
| Traversing | O(n) | O(1) |
| Deletion of a node | O(n) | O(1) |
| Deletion of CSLL | O(1) | |

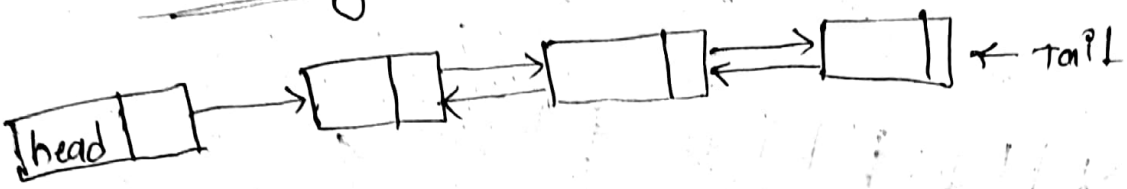
# Deletion Algorithm CSLL



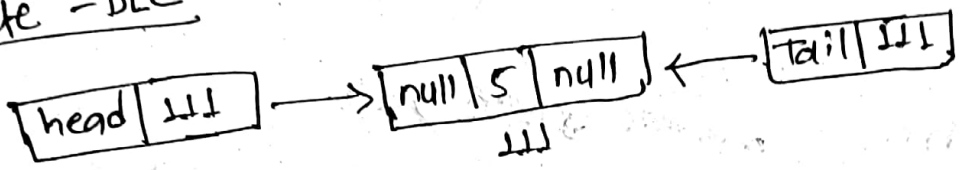

Method in pc!

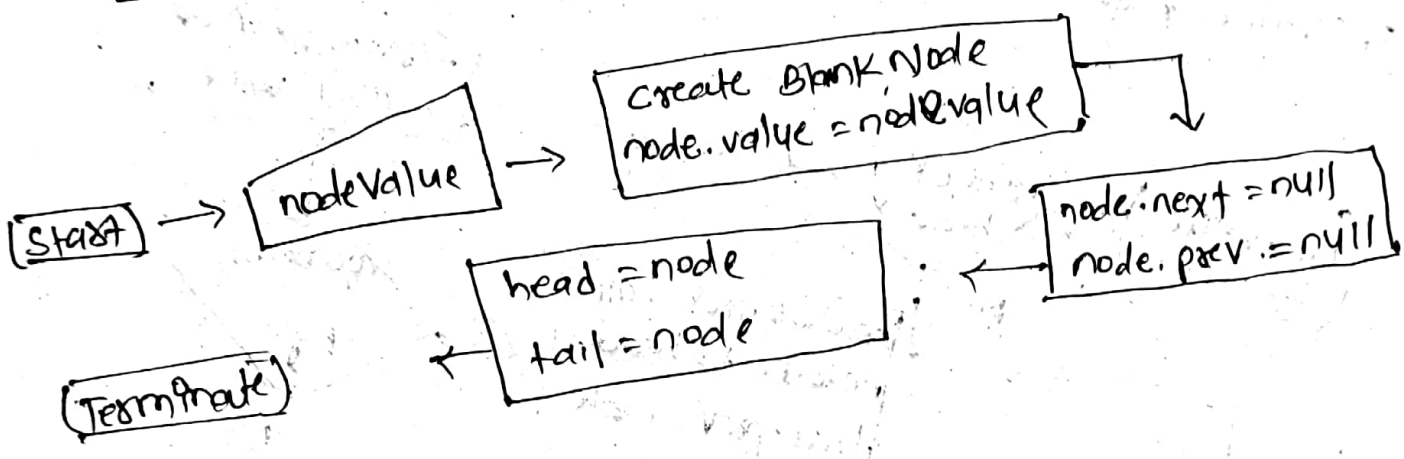
## Delete the entire CSLL

# 3. Doubly Linked list



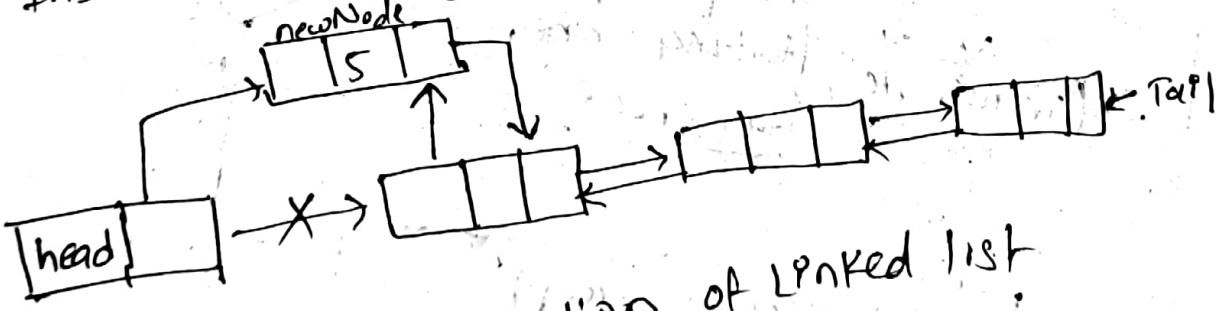

## 1. Create - DLL



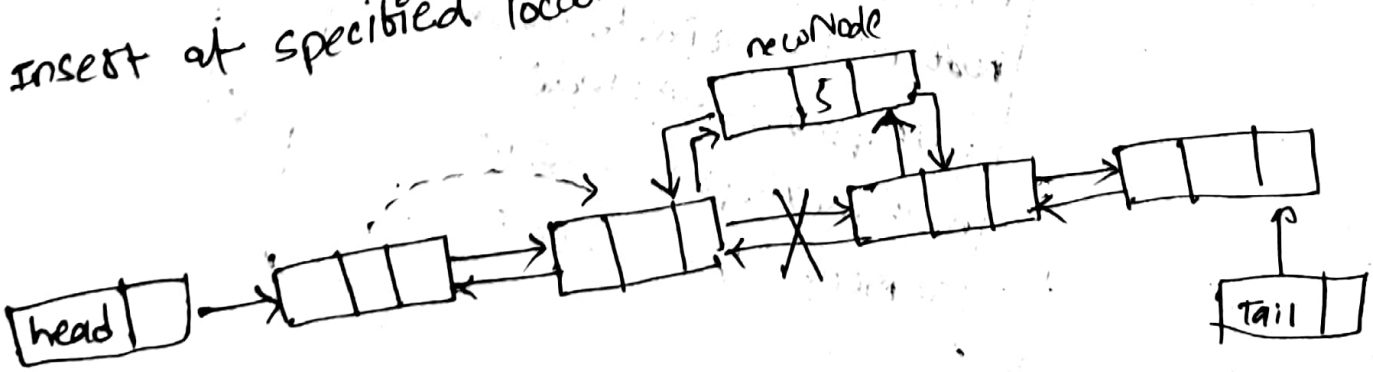




## 2. Insertion-DLL

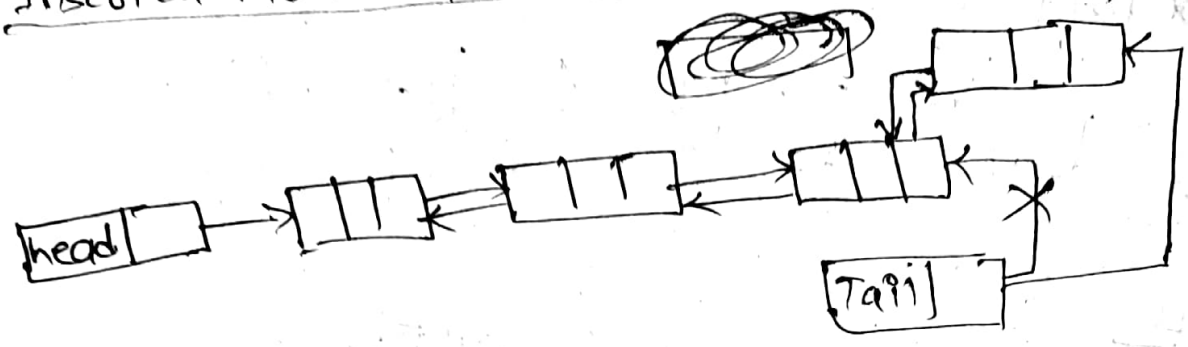
(a) Insert at the beginning of Linked list



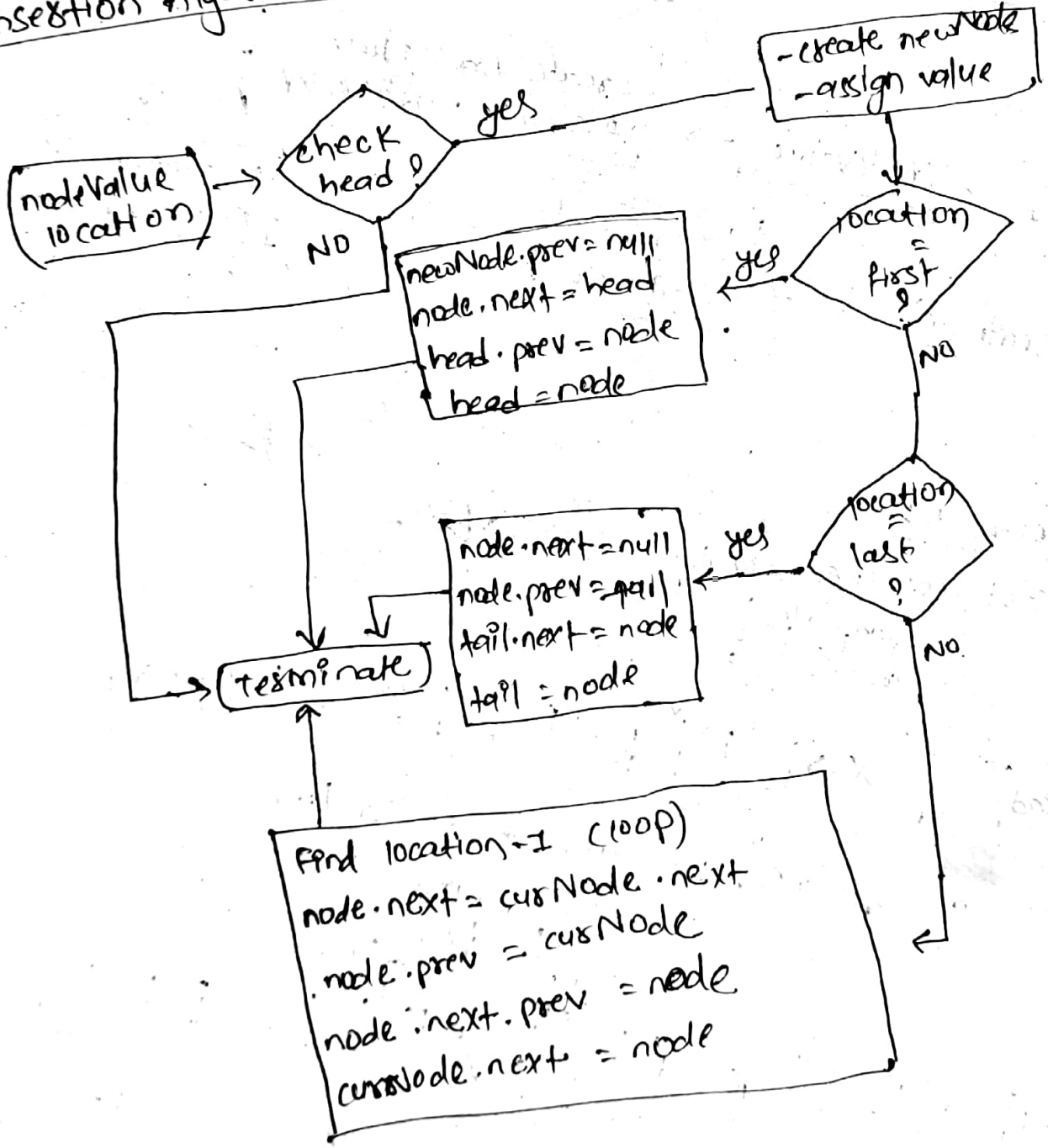

(b) Insert at specified location of Linked list



© Insert at the end of Linked list

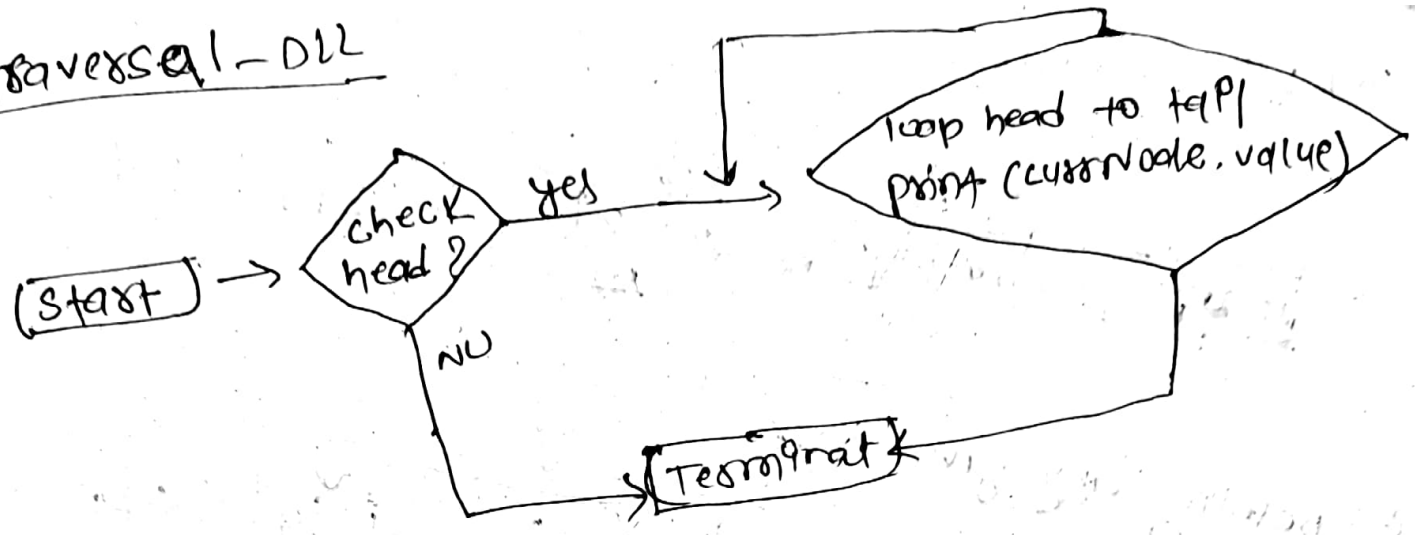



Insertion Algorithm — newNode ~~renamed~~ node




method in PC!

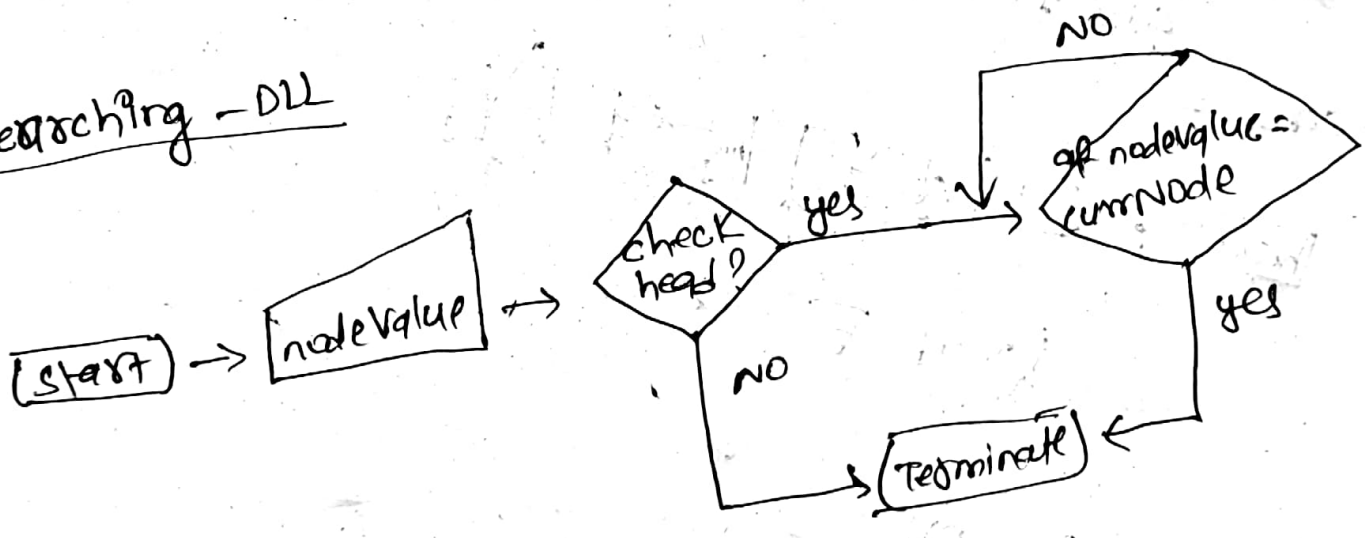
## 3. traversal - DLL




Reverse traversal - DLL

same → Just loop tail to head.

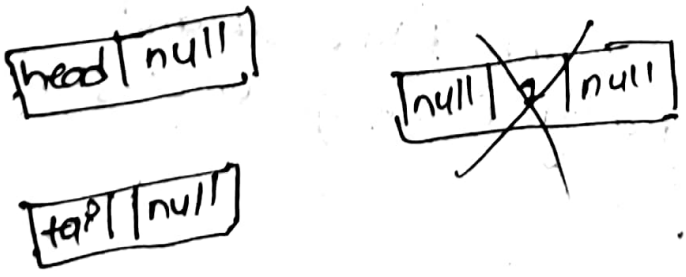
## 4. Searching - DLL



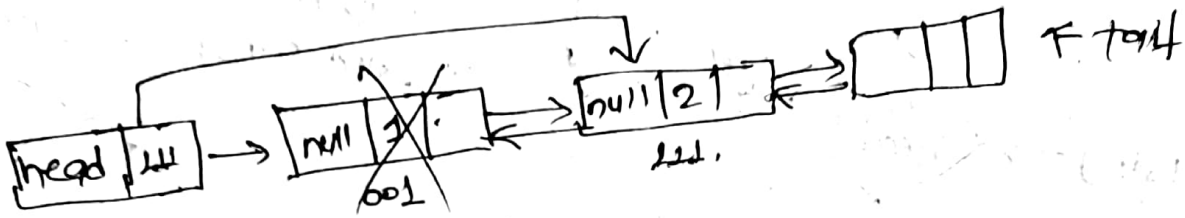

## 5. Deletion - DLL

(a) Deleting the first node

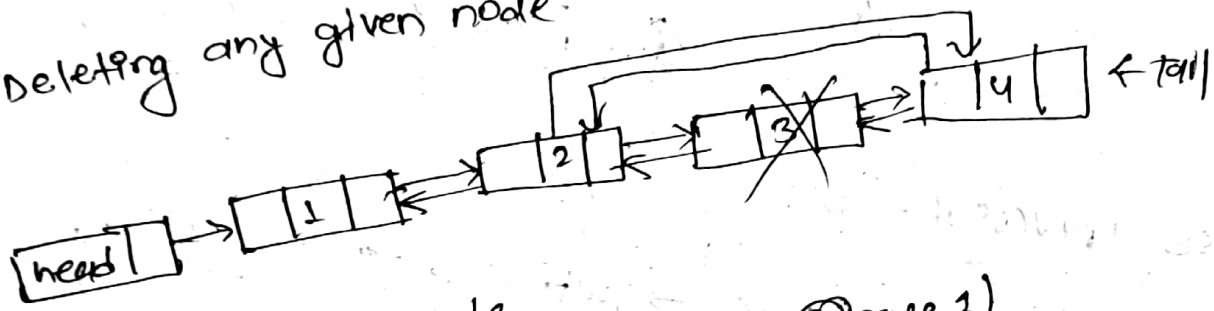
case - 1 : one node only



case 2 : more node

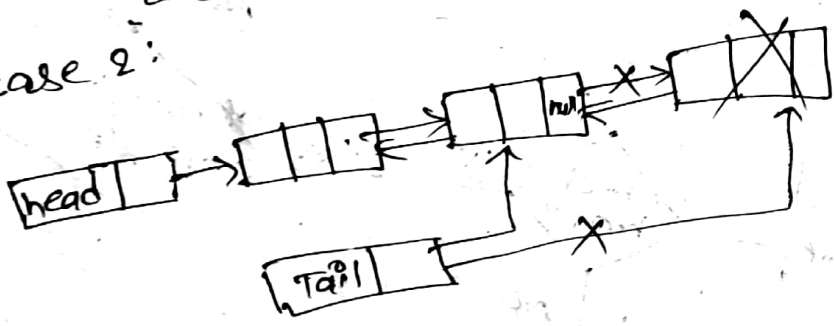



(b) Deleting any given node.

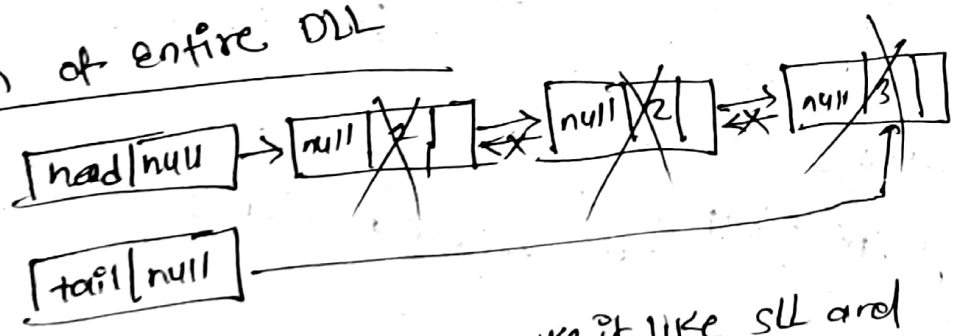



(c) Deleting the last node
case 1 : one node (same as (a) case 1)

case 2 :



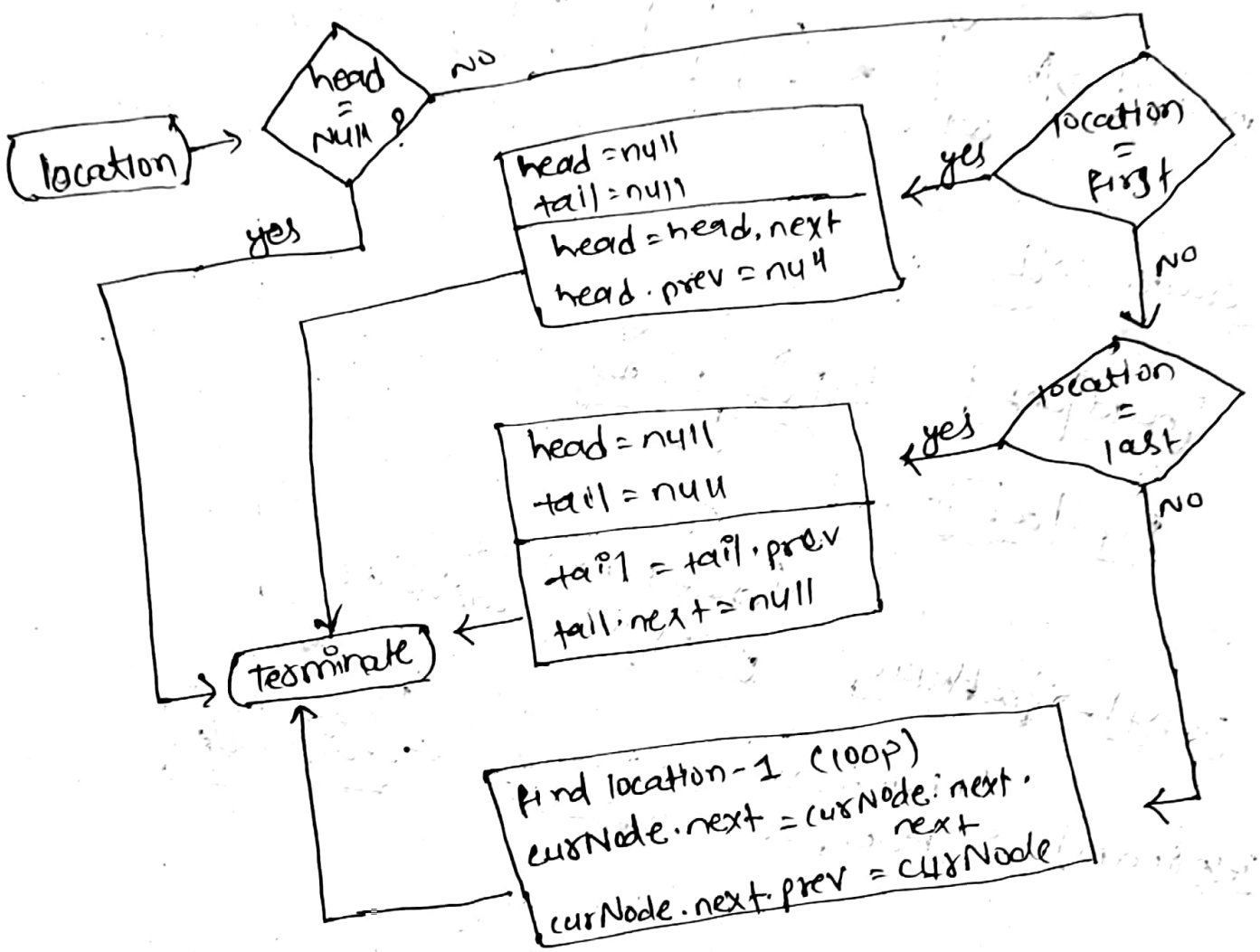

Deletion of entire DLL




① Delete each reference to make it like SLL and node can be deleted by Garbage collector

② head = tail = null.

# Deletion Algorithm - DLL

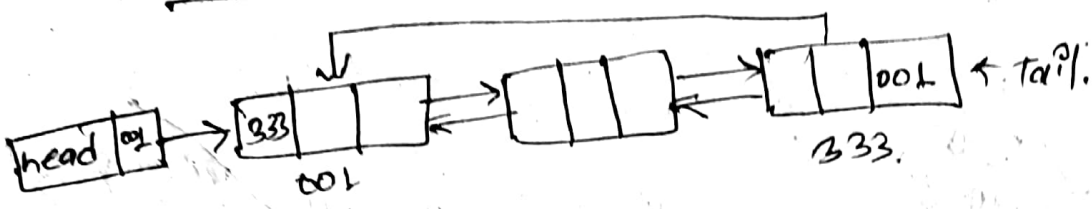



method in pg!

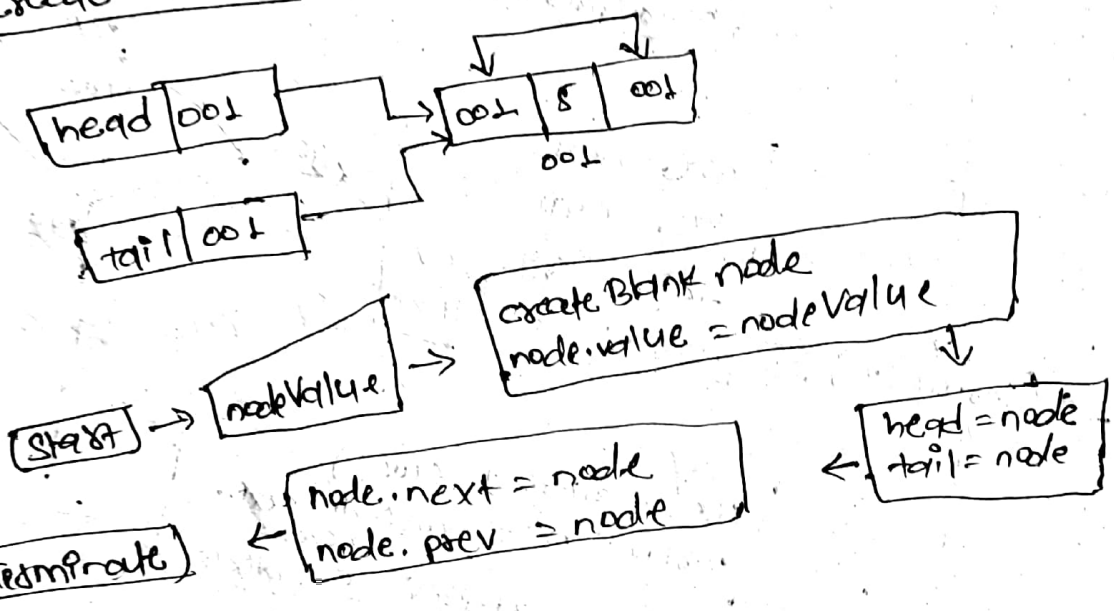
## Time and space complexity of DLL

| | Time complexity | space complexity |
|---|---|---|
| DLL creation | O(1) | O(1) |
| Insertion | O(n) | O(1) |
| Searching | O(n) | O(1) |
| Traversing (forward, backward) | O(n) | O(1) |
| Deletion of a Node | O(n) | O(1) |
| Deletion of DLL | O(n) | |

# 4. Circular Doubly linked list

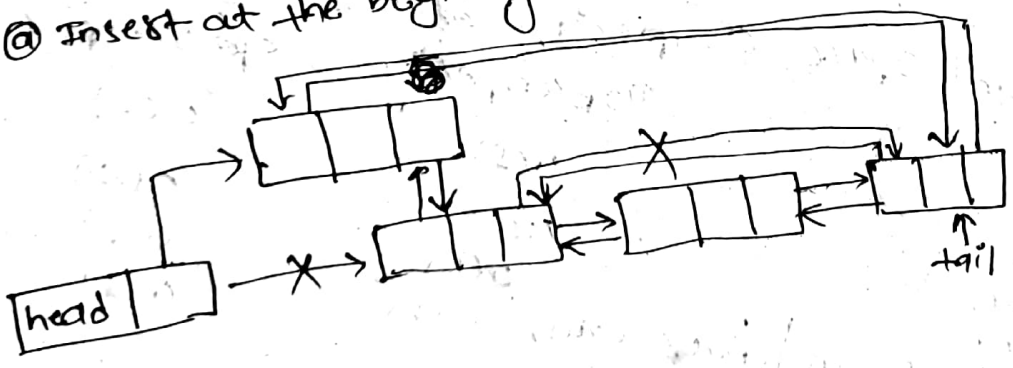


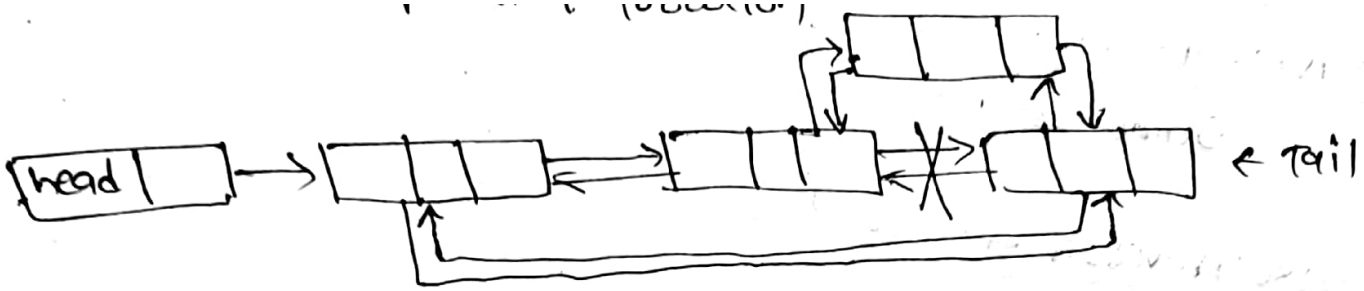

## 1. Create CDLL



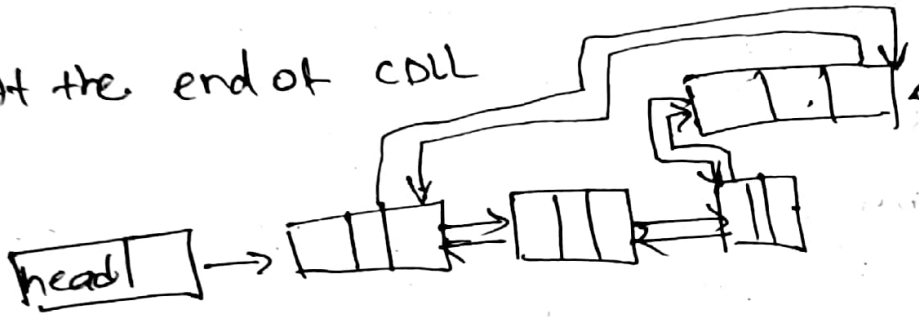

## 2. Insertion - CDLL

@ Insert at the begining of linked list

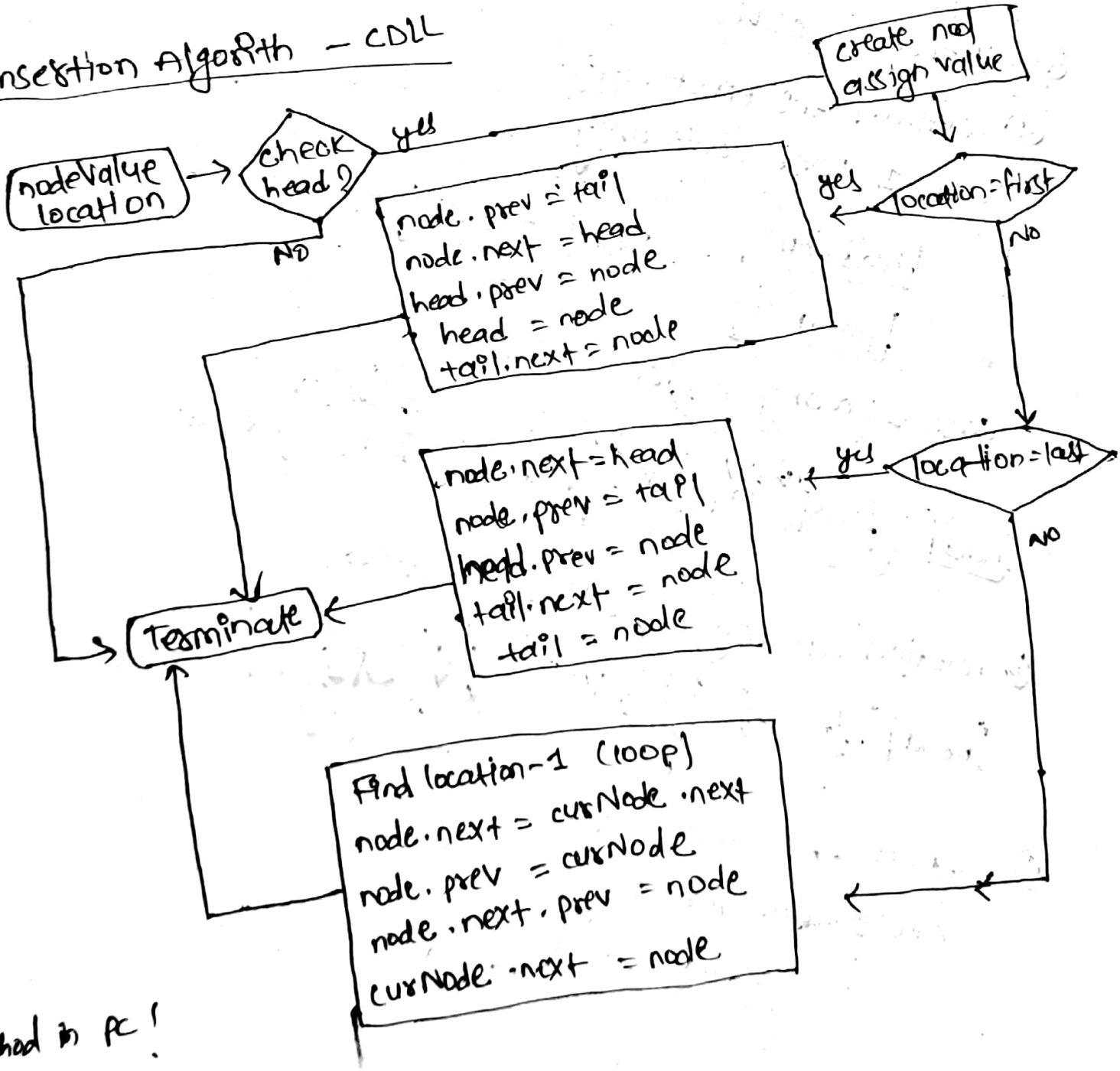





(6) At the end of CDLL




## Insertion Algorith – CDLL




Method in PC!

3. Traversal - CDLL

Same as DLL.

4. Reverse Traversal - CDLL

same as DLL

5. Search node - CDLL

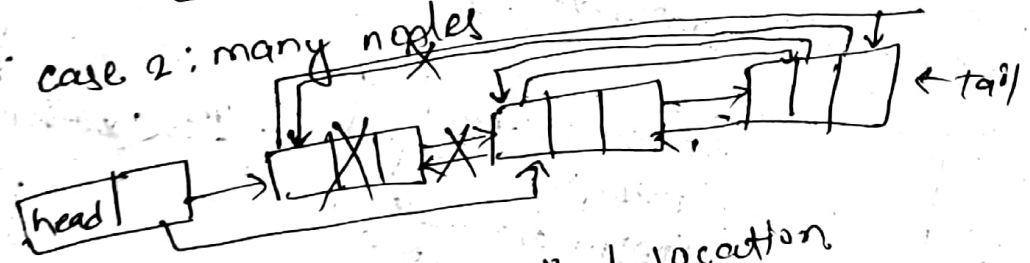
same as DLL..

6. Deletion - CDLL

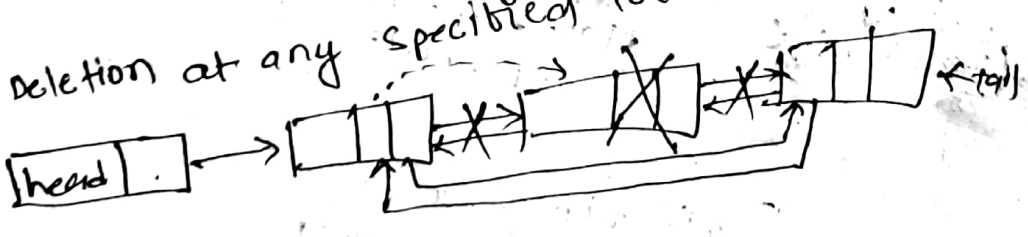
(a) Deletion the first node

case 1 : one node only


case 2 : many nodes




(b) Deletion at any specified location

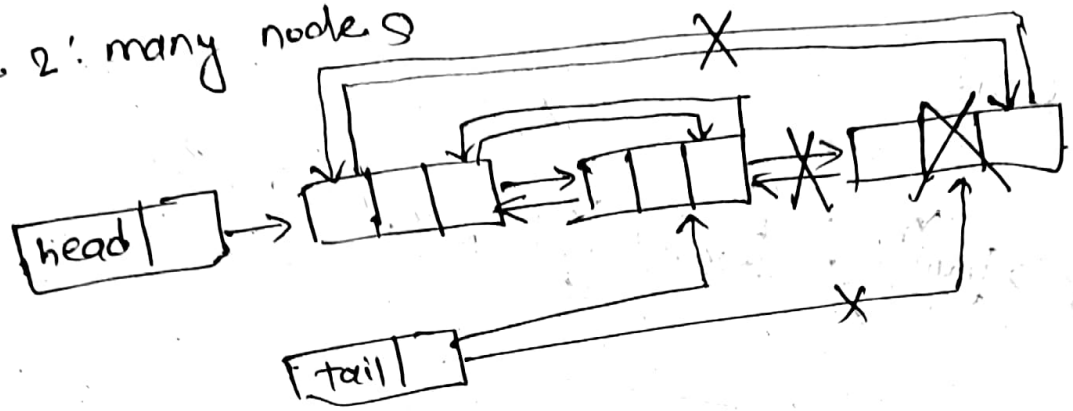



(c) Deletion the last node

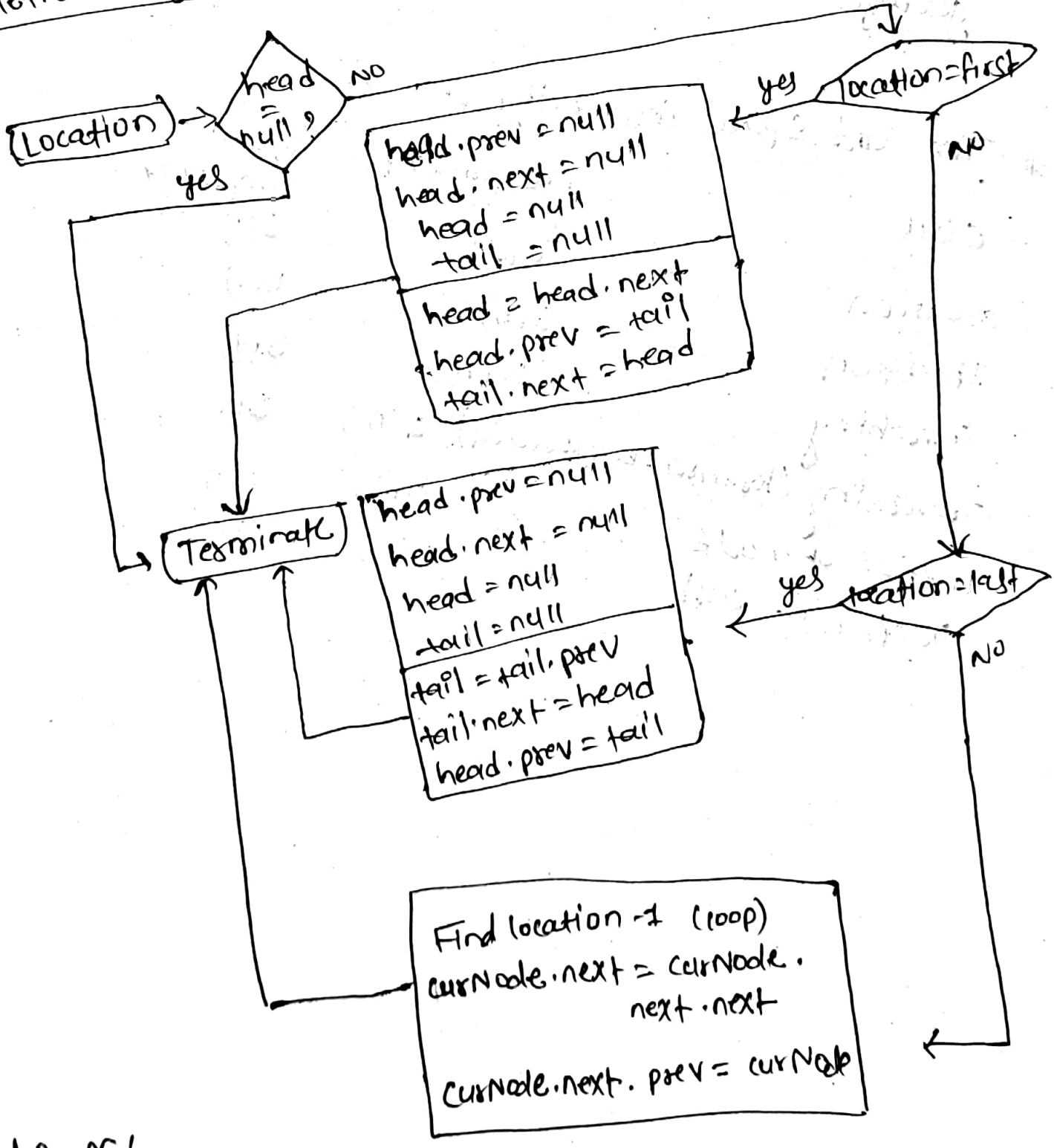
case 1 : one node only

Same as above ✓

case 2: many nodes

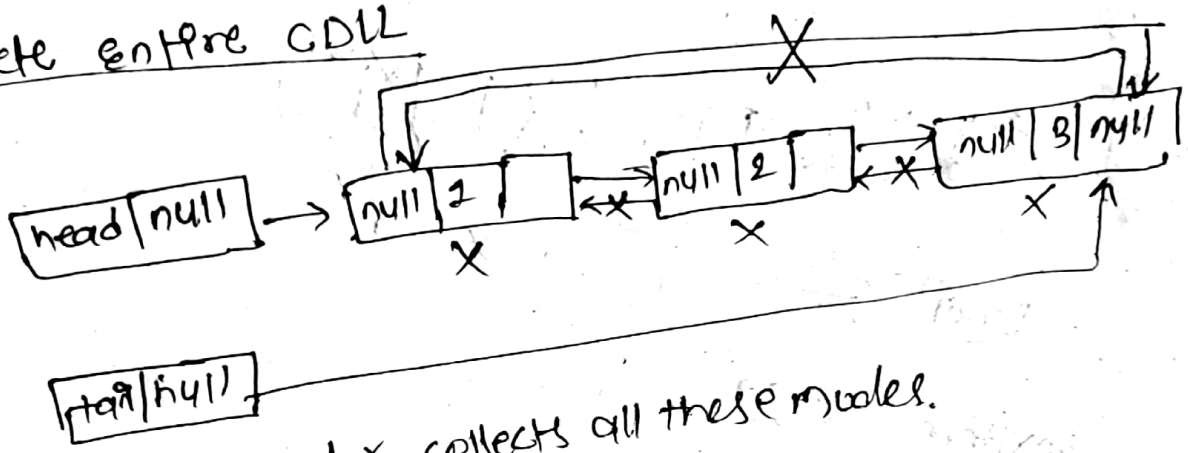



## Deletion Algorithm – CDLL




method in PC!

## Delete entire CDLL




garbage collector collects all these nodes.

## * Time and Space Complexity of CDLL

| | Time complexity | space complexity |
|---|---|---|
| CDLL | O(1) | O(1) |
| creation | O(n) | O(1) |
| Insertion | O(n) | O(1) |
| searching | O(n) | O(1) |
| Traversing (forward, backward) | O(n) | O(1) |
| Deletion of node | O(n) | O(1) |
| Deletion of CDLL | | |

# Time complexity of Array Vs Linked List

| Operation | Array | Linked list |
|---|---|---|
| creation | O(1) | O(1) |
| Insertion at first position | O(1) | O(1) |
| Insertion at last position | O(1) | O(1) |
| Insertion at nth position | O(1) | O(n) |
| Searching unsorted data | O(n) | O(n) |
| Searching sorted data | O(logn) | O(n) |
| Traversing | O(n) | O(n) |
| Deletion at 1st position | O(1) | O(1) |
| " " last " | O(1) | O(n)/O(1) |
| " " nth position | O(1) | O(n) |
| Deletion of array/linked list | O(1) | O(n)/O(1) |
| Access nth element | O(1) | O(n) |