CIRCULAR LINKED LIST

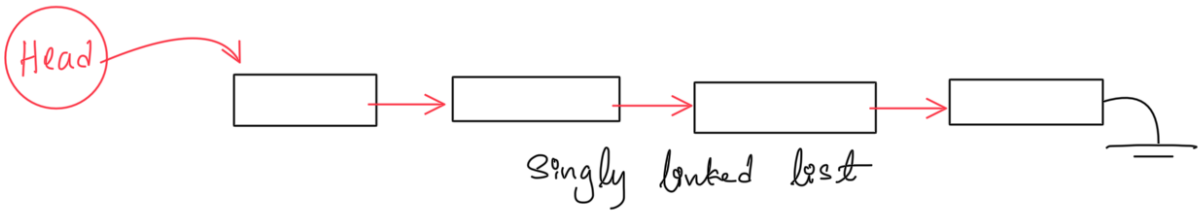
DAY 43
22/08/2025

What circular List is ?

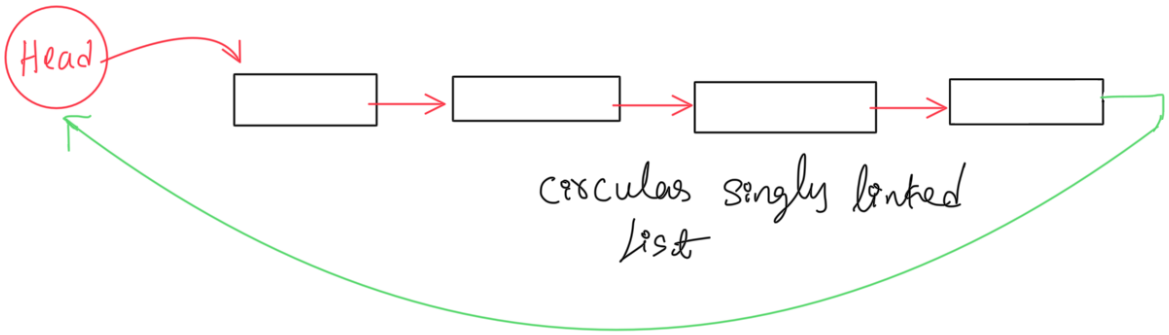
A circular Linked List is a variation of a linked List in which the lastNode points Back to the first node instead of pointing to null.

In a Singly Linked list last node's next points back to the head. (Becomes singly circular list)

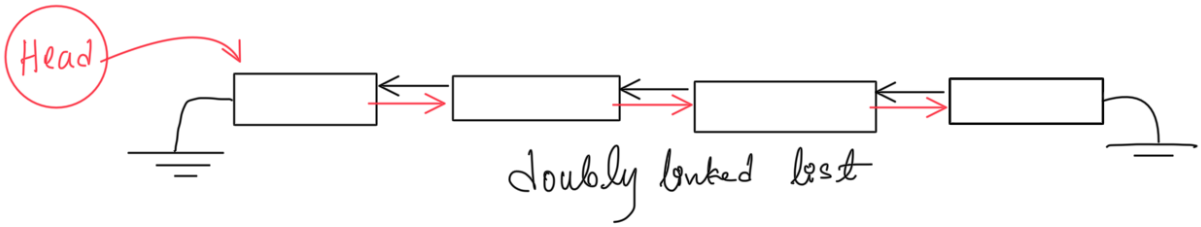
In doubly linked list last node's next points back to Head and Head.prev points Back to the lastNode



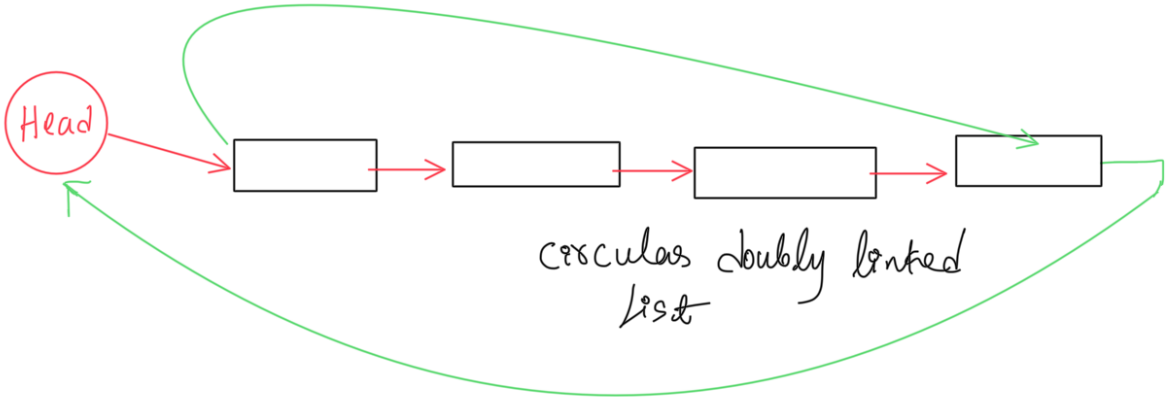
Singly linked list



circulas singly linked List



doubly linked list



circulas doubly linked List

🔵 What is a Circular List?

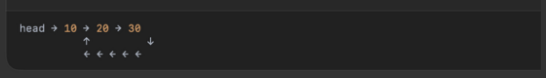
A **Circular Linked List (CLL)** is a variation of a linked list in which the **last node points back to the first node** instead of pointing to `null`.

- In a **Singly Circular Linked List (SCLL)** → last node's `next` points back to the head.
- In a **Doubly Circular Linked List (DCLL)** → last node's `next` points to head **and** head's `prev` points back to the last node.

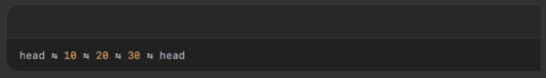
So, the list forms a **circle**, and you can traverse it infinitely.

🔵 Types of Circular Lists

1. **Singly Circular Linked List (SCLL)**
   - Each node has `data` and `next`.
   - `last.next = head` (circular link).
   - Example traversal: `head → node2 → node3 → … → last → head → …`



2. **Doubly Circular Linked List (DCLL)**
   - Each node has `data`, `next`, and `prev`.
   - `last.next = head` and `head.prev = last`.
   - Allows traversal in **both directions**.



🔵 Why Circular List? (Advantages / Use-Cases)

1. **Efficient Traversal**
   - You can go around the list starting from any node.
   - No need to go back to the head manually.
2. **Better for Circular Queues**
   - Useful when implementing **round-robin scheduling** (like CPU process scheduling).
   - Example: Each process gets a fixed time slice, then the scheduler moves to the next process in a circle.
3. **Continuous Navigation**
   - Ideal for applications like **media players (playlist loop), carousel UI components, traffic systems**, etc.
4. **Insertion efficiency**
   - Inserting at the beginning or end becomes easier, because you don't need to traverse to find the last node every time (in some cases you maintain a `tail` pointer).
5. **Memory Utilization**
   - No null links, so slightly less memory overhead compared to standard linked lists.

🔑 Quick Comparison

Feature	Singly Circular LL	Doubly Circular LL
Links per node	1 (`next`)	2 (`next`, `prev`)
Traverse direction	Forward only	Forward & Backward
Complexity	Simple	More complex
Use cases	Queues, round-robin	Playlists, Undo/Redo

🔵 Real Usage of Circular Linked Lists in Industry

1. Operating Systems (Round-Robin Scheduling)

- **Problem:** CPU scheduling where each process gets an equal time slice.
- **Why CLL?** The scheduler cycles through processes in a loop, and after the last process, it immediately returns to the first without extra checks.
- **Example:** Linux process scheduler, embedded RTOS schedulers.

2. Networking (Token Ring, Polling)

- **Problem:** In **token ring networks** or resource polling systems, nodes must be visited in a circular fashion.
- **Why CLL?** Perfect for looping through network nodes continuously.
- **Example:** Network packet routing in legacy token ring protocols, polling devices in IoT/embedded controllers.

3. Multiplayer Games / Simulations

- **Problem:** Players or entities take turns (like cards, board games, or simulations).
- **Why CLL?** Makes turn-based rotation simple — after the last player, you automatically move to the first.
- **Example:** Online board games, gaming servers, AI agent simulations.

4. Music / Media Players (Playlists)

- **Problem:** Repeat playlists or "shuffle loop."
- **Why CLL?** Once you reach the end of a playlist, CLL loops back to the first song naturally.
- **Example:** Spotify, VLC, YouTube "loop mode."

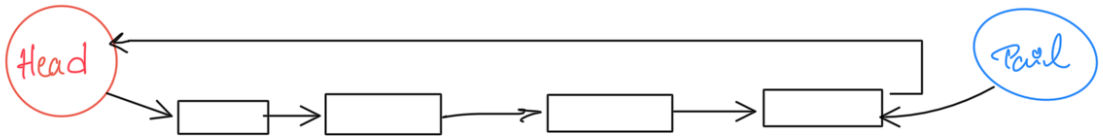
5. Undo/Redo Features

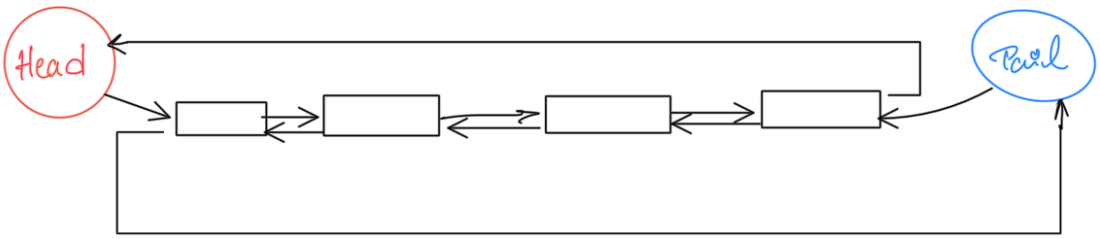
- **Problem:** In editors (text, graphics, IDEs), users may cycle through a history of actions.
- **Why CLL?** Doubly circular lists allow forward and backward traversal of states seamlessly.
- **Example:** Microsoft Word, Photoshop, Eclipse/IntelliJ undo/redo stack.

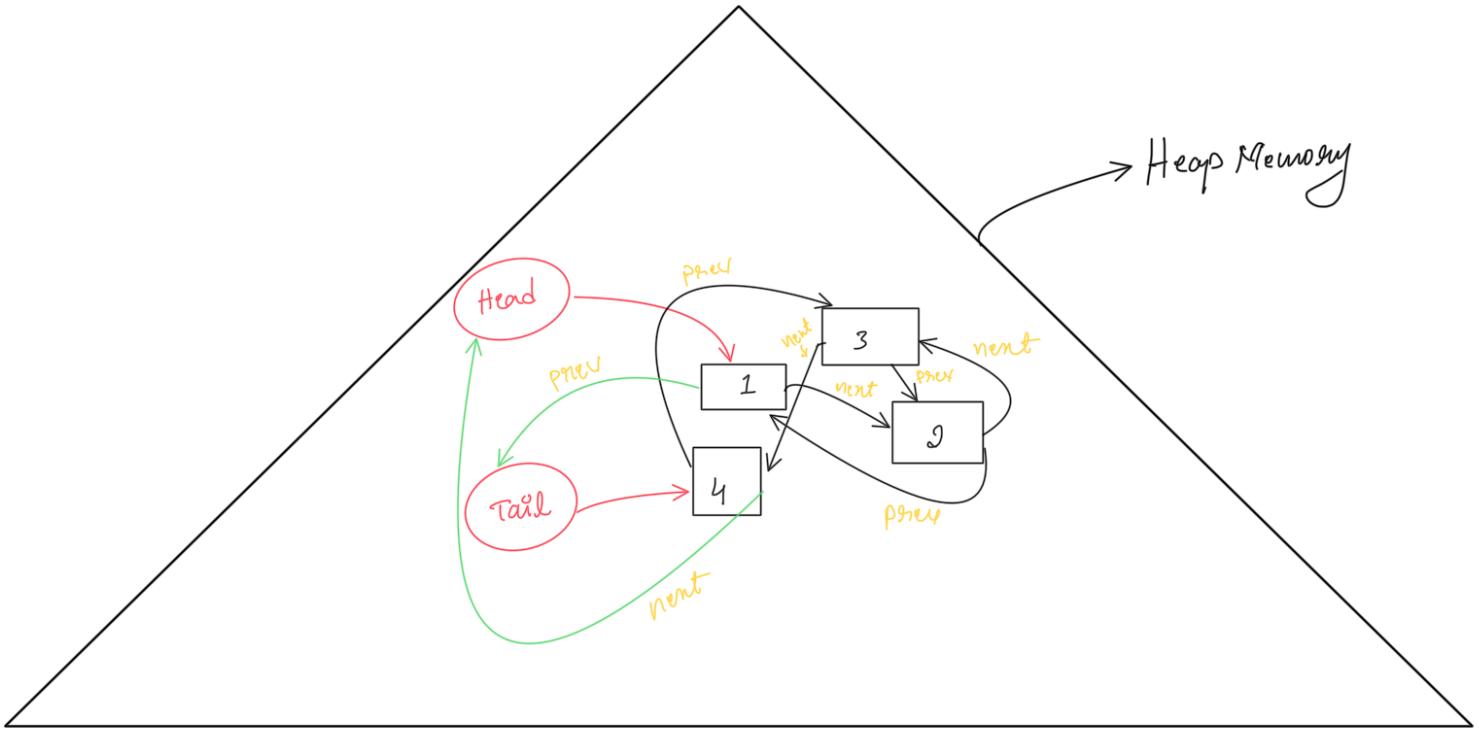
6. Circular Buffers / Queues

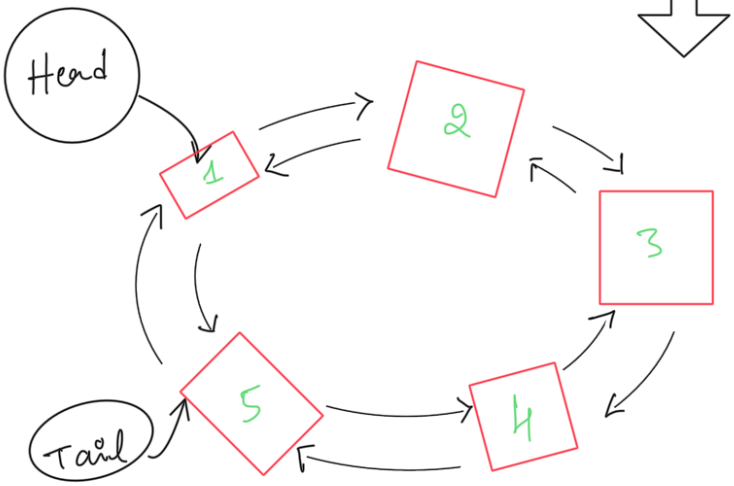
- **Problem:** Fixed-size buffers (like in streaming, logging, or hardware drivers).
- **Why CLL?** Can reuse memory efficiently by looping back when the buffer is full.
- **Example:**
  - Network routers (packet queues)
  - Real-time logging systems
  - Audio/video streaming buffers

We can use Tail Reference to last node of Circular Linked list.



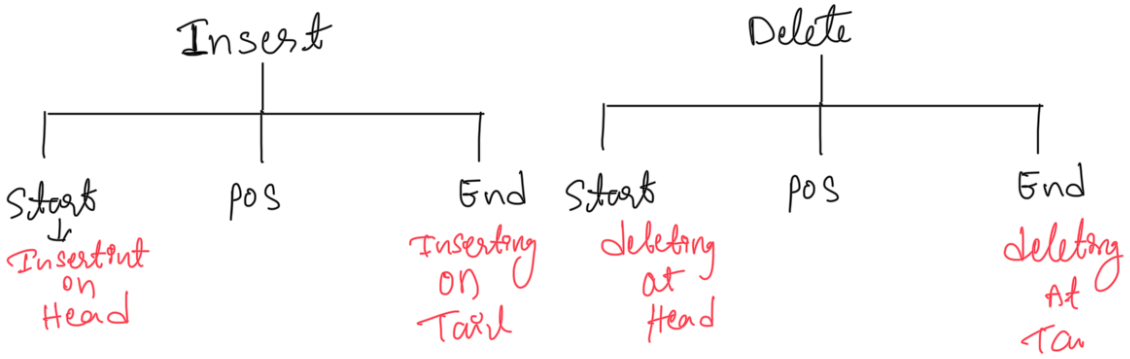






Visual Representation of a Circular Doubly linked list

Operations performed on Circular Linked List



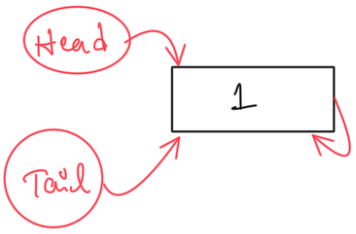
Searching & printing

Scenarios

(1) Empty List

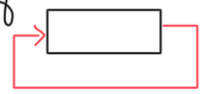
Head = tail = Null

(2) Single Node



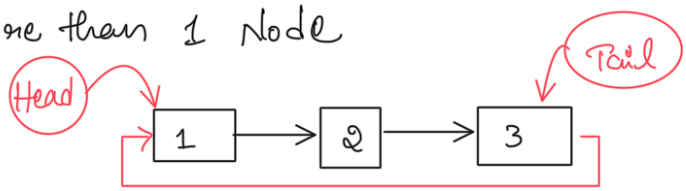
Head = Tail

The Node points to itself

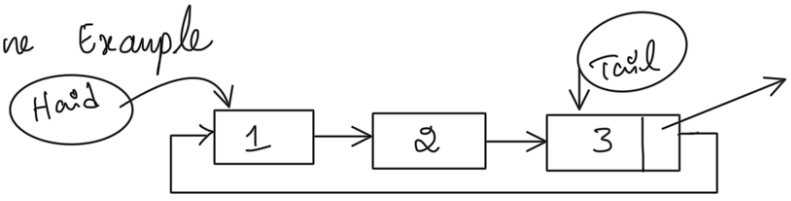


loop / circular

(3) More than 1 Node



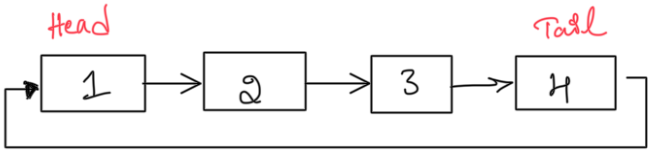
Let take one Example



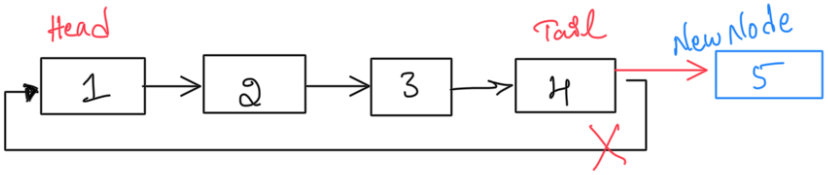
Tail.next pointing to Head

Insert operation At End / Front (Both are Same)

Insert At End

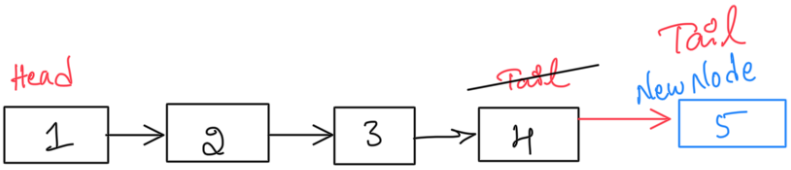


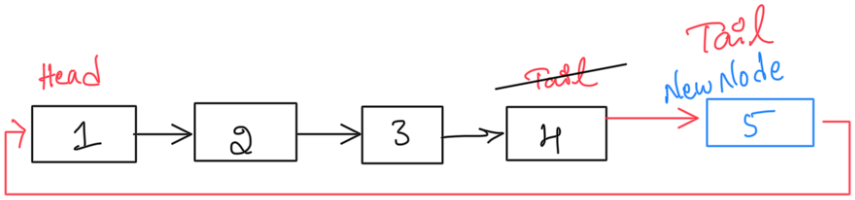
Tail.next = New node
Tail = New Node
Tail.next = Head



Tail.next = New Node

Tail = New Node



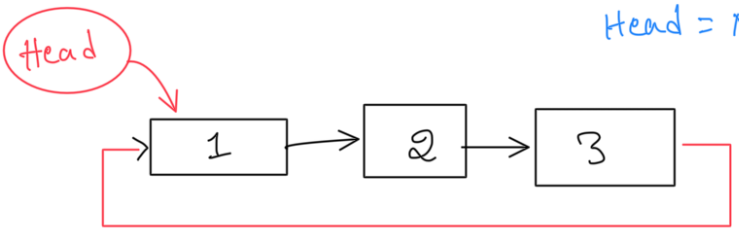


Tail.next = Head

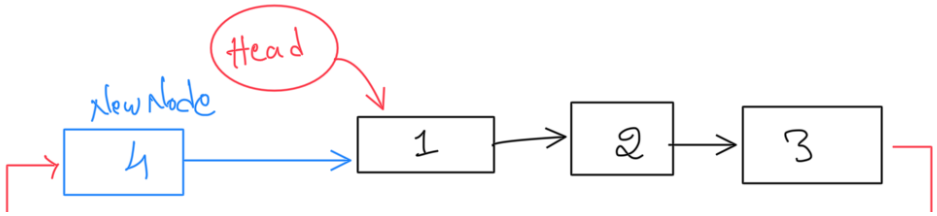
NOTE: Insert At the End Tail is updated

Insert At Front

New Node.next = Head
Head = New Node



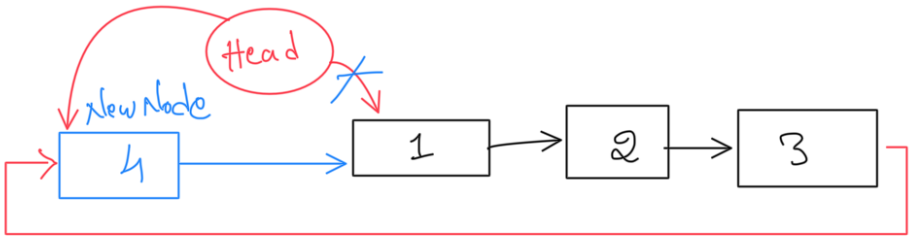
New Node.next = Head



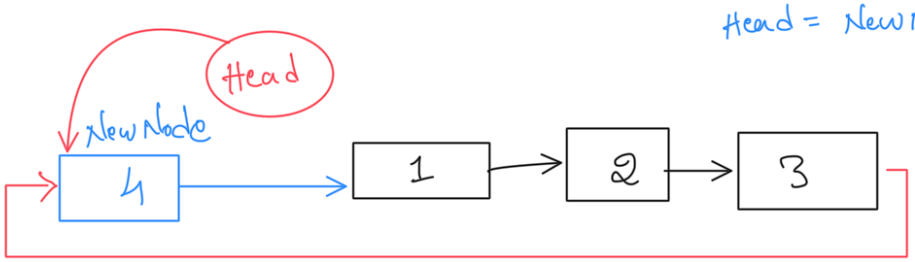
Tail.next

Always point to Head

Head = NewNode

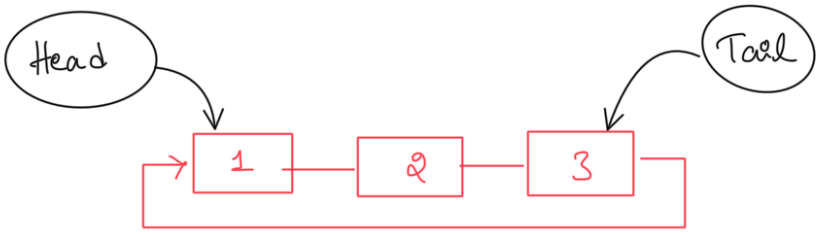


Head = NewNode



NOTE:
Tail pointer remain unTouched
Head is updated

Looping / Searching



use one Variable called curNode start from Head and traverse till tailNode.next (curNode.next != Head)

curNode = tail.next

while (curNode.next != tail.next) {

}

-Start from Head
Traverse till Tail

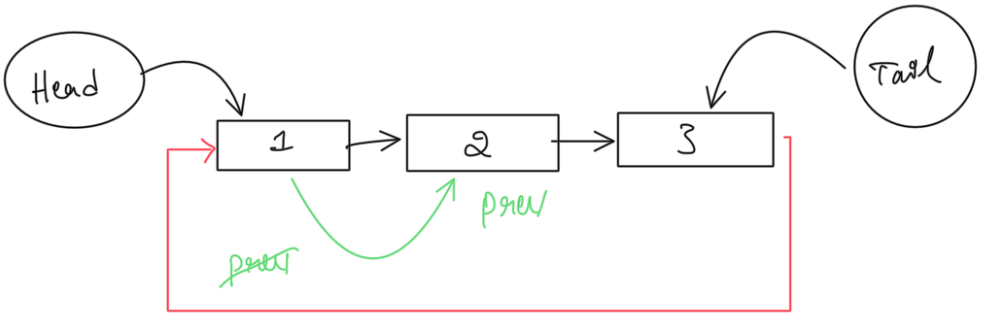
OR we can use a do while loop

```java
Dnode curNode = head;
if (curNode != null) {
    do {
        System.out.print(curNode.data + " ");
        curNode = curNode.next;
    } while (curNode != head);  // stop when back at start
}
```

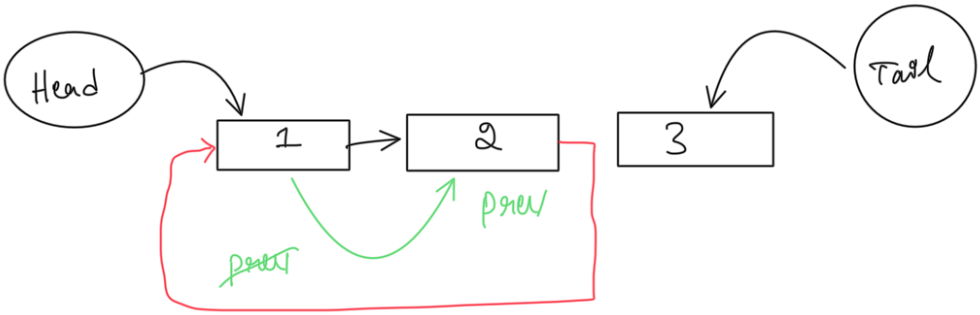
Delete At Tail

Traverse till last prev Node of Tail and update prevNode.next to Head, and tail = prevNode.

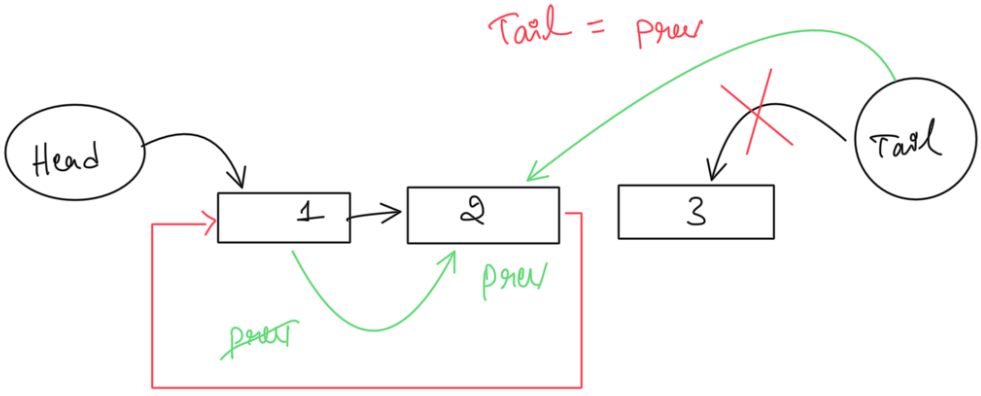
prevNode.next = Head
Tail = prevNode



prev.next = Head



prev.next = Head

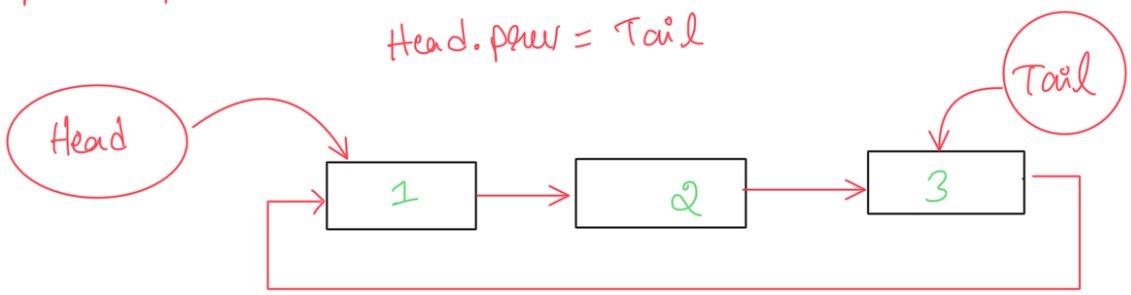


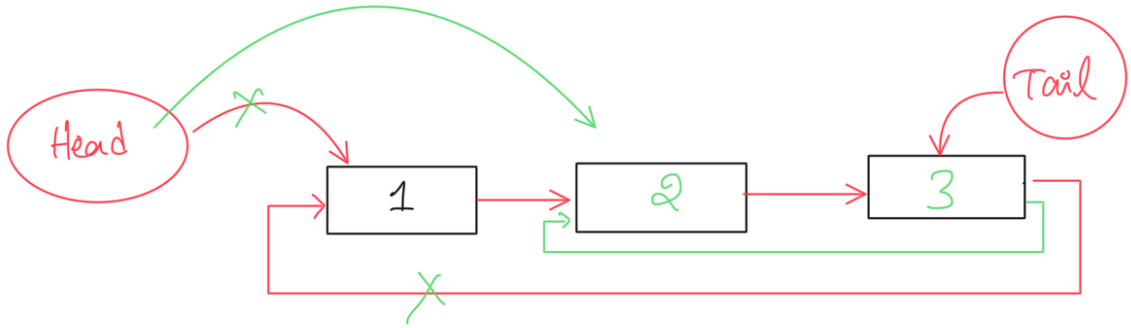
prev.next = Head

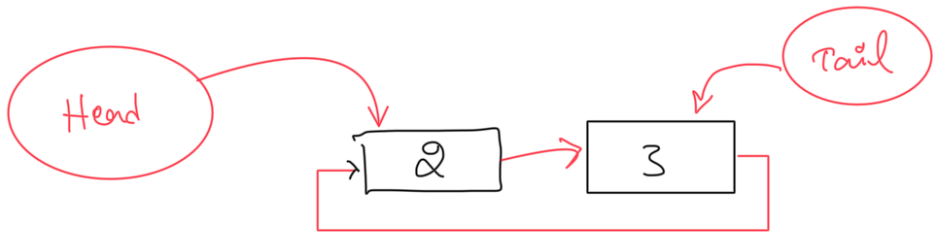
DELETE AT HEAD

Tail.next = Head

Simple Step Head = Head.next or tail.next = tail.next.next
Head.prev = Tail







NOTE : Dont forgot to Handle Corner Cases and Size updations (after deletion / insertion)