Experiment No. 4

Aim:

To design and implement a C program using Singly Linked List (SLL) to maintain the membership records of the Pinnacle Club, including operations like insertion, deletion, counting, display, and concatenation of lists.

Objective:

- To understand the application of Singly Linked Lists in real-world scenarios.
- To implement insertion and deletion operations at specific positions (President, Secretary, and general members).
- To manage dynamic records of students using linked list nodes (PRN and Name).
- To demonstrate list concatenation to merge two divisions' club member lists.
- To compute the total number of members in the club.

Problem Statement:

The Department of Computer Engineering has a student's club named **"Pinnacle Club"**.

- Membership is open to students of Second, Third, and Final Year.
- Students can join or cancel membership on request.
- The **first node** in the list is reserved for the **President**.
- The **last node** is reserved for the **Secretary**.
- Each node stores the **PRN** (Permanent Registration Number) and **Name** of the student.

Write a program to:
a) Add and delete members (including adding/deleting President or Secretary).
b) Compute the total number of members in the club.
c) Display the list of members.

Theory:

- **Linked List** is a dynamic data structure consisting of nodes.
- Each **node** contains:
  - **Data field** → PRN, Name
  - **Pointer field** → Address of next node

- Operations required:

1. **Insertion** → Add President, Secretary, or General Member.
2. **Deletion** → Remove President, Secretary, or General Member.
3. **Counting** → Traverse and count nodes.
4. **Display** → Print member list.

Algorithm:

1. Add Member

- Create new node with PRN and Name.
- If President → Insert at beginning.
- If Secretary → Insert at end.
- If General Member → Insert between nodes.

2. Delete Member

- If President → Delete first node.
- If Secretary → Delete last node.
- If Member → Search node by PRN, then delete.

3. Count Members

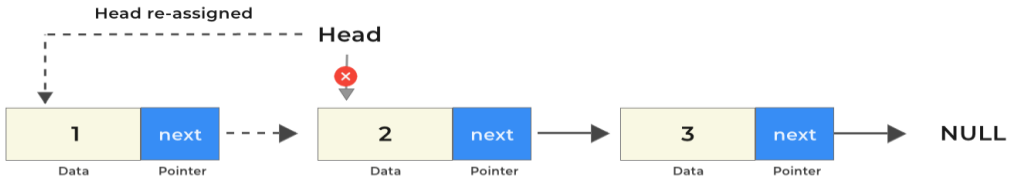
- Initialize count = 0.
- Traverse list and increment count for each node.
- Display count.

4. Display Members

- Traverse list from head to end.
- Print PRN and Name.

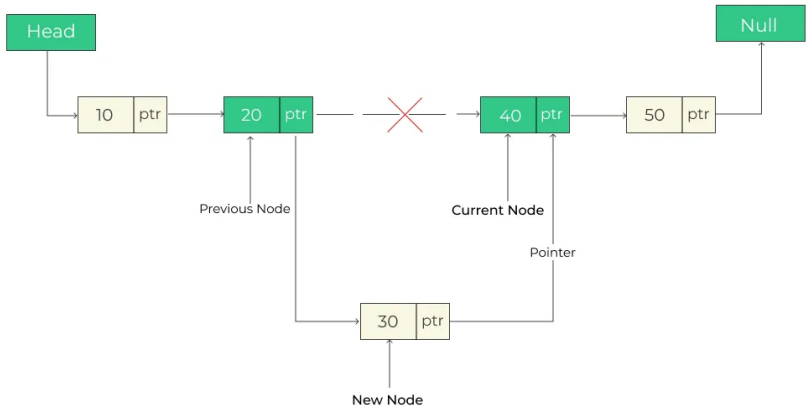
Insertion in the beginning of linked list:

- To insert a new node at the front, we create a new node and point its next reference to the current head of the linked list. Then, we update the head to be this new node. This operation is efficient because it only requires adjusting a few pointers.



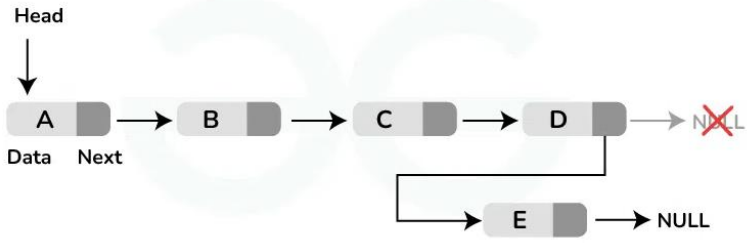
Insert a Node after a Given Node in Linked List

- If we want to insert a new node after a specific node, we first locate that node. Once we find it, we set the new node's next reference to point to the node that follows the given node. Then, we update the given node's next to point to the new node. This requires traversing the list to find the specified node.



Insert a Node at the End of Linked List

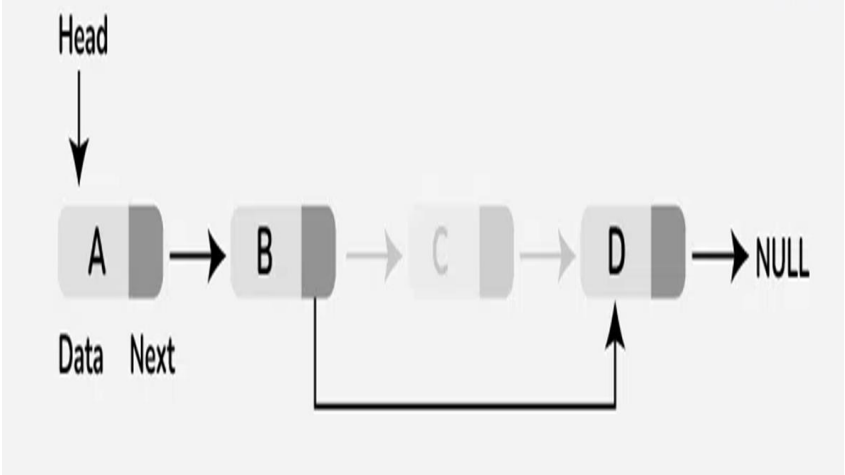
Inserting at the end involves traversing the entire list until we reach the last node. We then set the last node's next reference to point to the new node, making the new node the last element in the list.



Deleting first node of linked list:

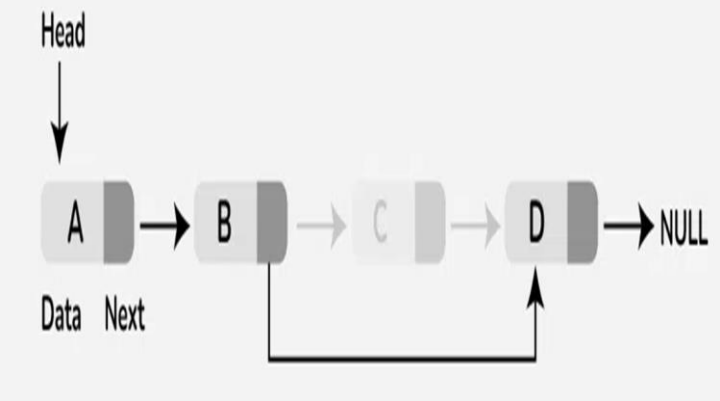
Step-by-Step Approach:

- Check if the list is empty: If the head is NULL, the list is empty, and there's nothing to delete.
- Update the head pointer: Set the head to the second node (head = head->next).
- Delete the original head node: The original head node is now unreferenced, and it can be freed/deleted if necessary



Deleting specific node:

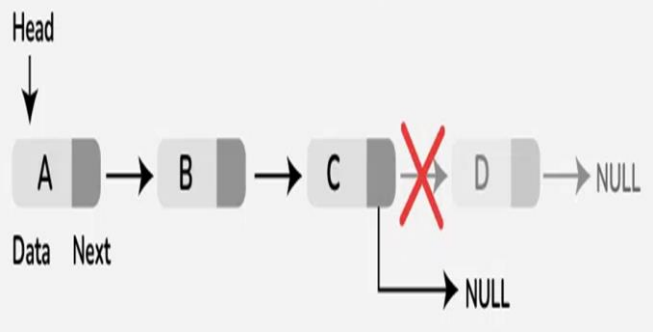
- Check if the position is valid: If the position is out of bounds (greater than the length of the list), return an error or handle appropriately.
- Traverse the list to find the node just before the one to be deleted: Start from the head and move through the list until reaching the node at position n-1 (one before the target position).
- Update the next pointer: Set the next pointer of the $(n-1)^{th}$ node to point to the node after the target node (node_to_delete->next).
- Delete the target node: The node to be deleted is now unreferenced, and in languages like C++ or Java, it can be safely deallocated.



Deletion of last node:

- Check if the list is empty: If the head is NULL, the list is empty, and there's nothing to delete.
- If the list has only one node: Simply set the head to NULL (the list becomes empty).
- Traverse the list to find the second-last node: Start from the head and iterate through the list until you reach the second-last node (where the next of the node is the last node).
- Update the next pointer of the second-last node: Set the second-last node's next to NULL (removing the link to the last node).
- Delete the last node: The last node is now

unreferenced and can be deleted or freed.



Input:

Note: Write input you are giving to your program

Output:

Note: Write output of your program

Conclusion:

Hence we have implemented all the operations like insertion, deletion, counting, and display on Singly Linked List successfully.