Linked List 1

Data Structures

Data structures are just a way to store and organise our data so that it can be used and retrieved as per our requirements. Using these can affect the efficiency of our algorithms to a greater extent. There are many data structures that we will be going through throughout the course, linked list is a part of them.

Introduction to linked list

Let's first discuss what are the disadvantages of using an array:

- Arrays have a fixed size that is required to be initialised at the time of declaration i.e., the addition of extra elements would throw an error (Index out of range) in C++.
- Arrays could store a maximum of 10^6 - 10^7 elements . If we try to store more elements, then the program might lead to memory overflow.
- Even deletion of elements from the array is an expensive task as it would require the complete shift of further elements by some positions to the left as the data is stored in the form of contiguous memory blocks.

In order to overcome these problems, we can use linked lists...

- A linked list is a linear data structure where each element is a separate object.
- Each element or node of a list is comprising of two items:
    - Data
    - Pointer(reference) to the next node.
- A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.
- The first node of a linked list is known as head.
- The last node of a linked list is known as tail.
- The last node has a reference to null.

Linked list class

```
class Node {
        public :
        int data;                       // to store the data stored
        Node *next;                     // to store the address of next pointer

        Node(int data) {
                this -> data = data;
                next = NULL;
        }
};
```

Note: The first node in the linked list is known as **Head** pointer and the last node is referenced as **Tail** pointer. We must never lose the address of the head pointer as it references the starting address of the linked list and is, if lost, would lead to losing of the list.

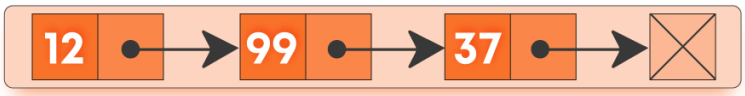
Printing of the linked list

To print the linked list, we will start traversing the list from the beginning of the list(head) until we reach the NULL pointer which will always be the tail pointer. Follow the code below:

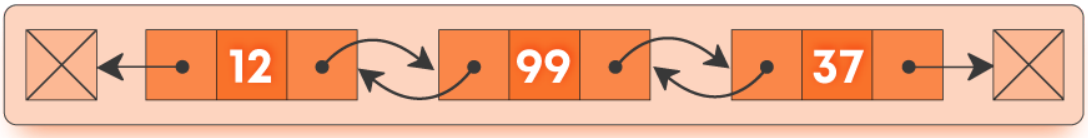
```
void print(Node *head) {
      Node *tmp = head;
      while(tmp != NULL) {
                  cout << tmp->data << " ";
                  tmp = tmp->next;
      }
      cout << endl;
}
```

There are generally three types of linked list:

- **Singly:** Each node contains only one link which points to the subsequent node in the list.



- **Doubly:** It's a two-way linked list as each node points not only to the next pointer but also to the previous pointer.



- **Circular:** There is no tail node i.e., the next field is never NULL and the next field for the last node points to the head node.

Let's now move onto all other operations like insertion, deletion, input in linked list...

Taking input in list

Refer the code below:

```
Node* takeInput() {
        int data;
        cin >> data;
        Node *head = NULL;
        Node *tail = NULL;
        while(data != -1) {                                    // -1 is used for terminating
                Node *newNode = new Node(data);
                if(head == NULL) {
                        head = newNode;
                        tail = newNode;
                }
                else {
                        tail -> next = newNode;
                        tail = tail -> next;
                        // OR
                        // tail = newNode;
                }

                cin >> data;
        }
        return head;
}
```

To take input in the user, we need to keep few things in the mind:

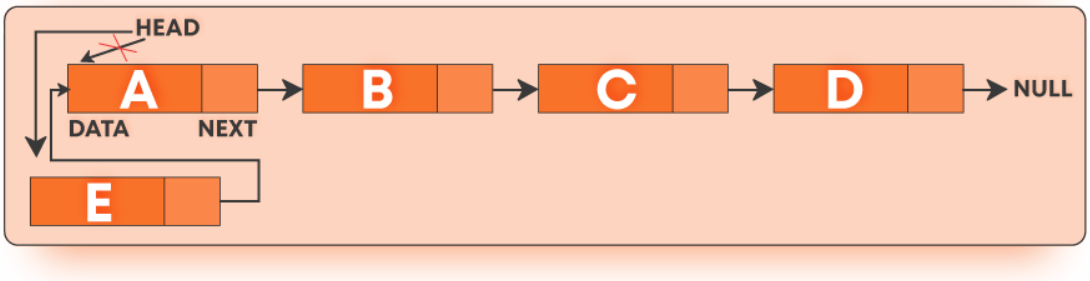
- Always use the first pointer as the head pointer.
- When initialising the new pointer the next pointer should always be referenced to NULL.
- The current node's next pointer should always point to the next node to connect the linked list.

Insertion of node

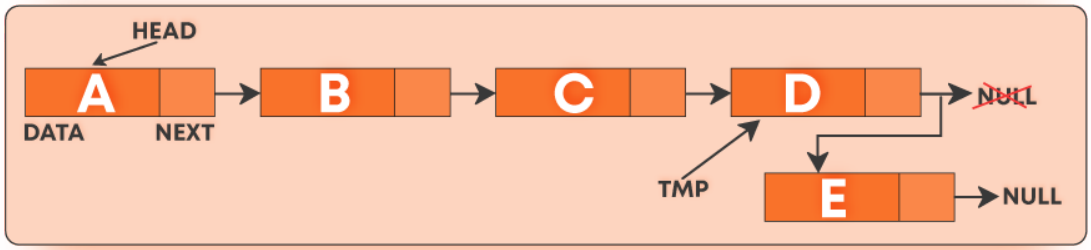
There are 3 cases:

- **Case 1: Insert node at the last**

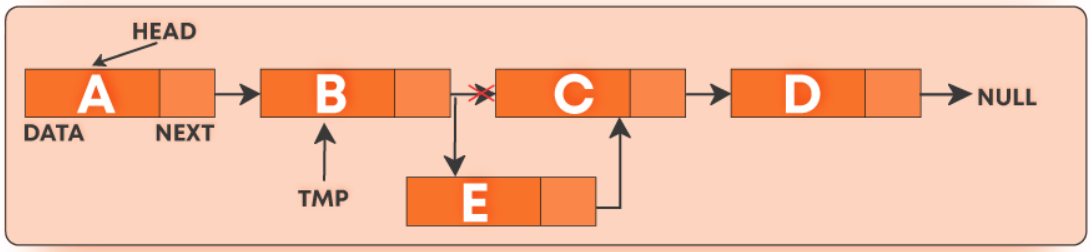
  This can be directly done by normal insertion as discussed above.



- **Case 2: Insert node at the beginning**
  - First-of-all store the head pointer in some other pointer.
  - Now, mark the new pointer as the head and store the previous head to the next pointer of the current head.
  - Update the new head.



- **Case 3: Insert node anywhere in the middle**
    - For this case, we always need to store the address of the previous pointer as well as the current pointer of the location at which new pointer is to be inserted.
    - Now let the new inserted pointer be curr. Point the previous pointer's next to curr and curr's next to the original pointer at the given location.
    - This way the new pointer will be inserted easily.



Let's now check out the code for all the above cases...

```
Node* insertNode(Node *head, int i, int data) {
        Node *newNode = new Node(data);
        int count = 0;
```

```
        Node *temp = head;

        if(i == 0) {                                        //Case 2
                newNode -> next = head;
                head = newNode;
                return head;
        }

        while(temp != NULL && count < i - 1) {              //Case 3
                temp = temp -> next;
                count++;
        }
        if(temp != NULL) {
                Node *a = temp -> next;
                temp -> next = newNode;
                newNode -> next = a;
        }
        return head;                    //Returns the new head pointer after insertion
}
```

Deletion of node

There are 2 cases:

- **Case 1: Deletion of the head pointer**

  In order to delete the head node, we can directly remove it from the linked list by pointing the head to the next.

- **Case 2: Deletion of any node in the list**

  In order to delete the node from the middle/last, we would need the previous pointer as well as the next pointer to the node to be deleted. Now directly point the previous pointer to the current node's next pointer.

Try to code it yourself and then check for the solution provided against the corresponding problem in the course curriculum.

Now, let's move on to the recursive approach for insertion and deletion of the node in a linked list.

Insert node recursively

Follow the steps below and try to implement it yourselves:

- If Head is null and position is not 0. Then exit it.
- If Head is null and position is 0. Then insert a new Node to the Head and exit it.
- If Head is not null and position is 0. Then the Head reference set to the new Node. Finally, a new Node set to the Head and exit it.
- If not, iterate until finding the Nth position or end.

For the code, refer to the Solution section of the problem...

Delete node recursively

Follow the steps below and try to implement it yourselves:

- If the node to be deleted is root, simply delete it.
- To delete a middle node, we must have a pointer to the node previous to the node to be deleted. So if the position is not zero, we run a loop position-1 times and get a pointer to the previous node.
- Now, simply point the previous node's next to the current node's next and delete the current node.

For the code, refer to the Solution section of the problem...

You will find more practice problems from other sources in the Linked List 2 module.