(Q1) - Linked list implementation in C++.

Code :

```
#include<iostream>

using namespace std;

class Node{
	public :
		int data;
		Node *next;
	//constructer
	Node(int data){
		this->data = data;
		this->next = NULL;
	}
	~Node(){
		int value = this->data;
		if(this->next != NULL){
			delete next;
			this->next = NULL;
		}
		cout<<"Memory is free"<<endl;
	}
};

//Node insert at head
void InsertAtHead(Node* &head, int d){
	Node* temp = new Node(d);
	temp->next = head;
	head = temp;
}

void InsertAtTail(Node* &tail, int d){
	Node* temp = new Node(d);
	tail->next = temp;
	tail = tail->next;
}

void InsertAtPosition(Node* &tail, Node* & head,int position, int d){
	//insert at start
	if(position ==1){
		InsertAtHead(head,d);
```

```
            return;
        }


        Node* temp = head;
        int c=1;

        while(c<position-1){
            temp = temp->next;
            c++;
        }

        //end tail
        if(temp->next == NULL){
            InsertAtTail(tail,d);
            return;
        }

        Node*  nodetoinsert = new Node(d);
        nodetoinsert->next = temp->next;
        temp->next = nodetoinsert;

}

//print list
void print(Node* &head){
        Node* temp = head;
        while(temp != NULL){
            cout<<temp->data<< " ";
            temp = temp->next;
        }
        cout<<endl;
}

void deleteNode(int position, Node* &head){
        if(position == 1){
            Node* temp = head;
            head = head->next;
            temp->next = NULL;
            delete temp;
        }
        else{
            Node* curr = head;
            Node* prev = NULL;
```

```
                int c=1;
                while(c< position){
                        prev = curr;
                        curr = curr->next;
                        c++;
                }
                prev->next = curr->next;
                curr->next = NULL;
                delete curr;

        }
}

int main(){
        int a,b,c,d,i;
        Node* node1 = new Node(10);
        cout<< node1->data <<endl;
        // head pointedd to node1
        Node* head = node1;
        Node* tail = node1;
        cout<<"Enter the numbers you want to insert at beginning"<<endl;
        cin>>a>>b;
        InsertAtHead(head,a);
        print(head);
        InsertAtHead(head,b);
        print(head);
        cout<<"Enter the numbers you want to insert at end"<<endl;
        cin>>c>>d;
        InsertAtTail(tail,c);
        print(head);
        InsertAtTail(tail,d);
        print(head);
        cout<<"Enter the numbers index you want the next number"<<endl;
        cin>>i;
        InsertAtPosition(tail,head,i,100);
        print(head);
        cout<<"Head "<<head->data <<endl;
        cout<<"Tail "<<tail->data <<endl;
        cout<<"Enter the indices you want to delete"<<endl;
        cin>>a>>b;
        deleteNode(a,head);
        print(head);
        deleteNode(b-1,head);
        print(head);
```

```
	cout<<"Head "<<head->data <<endl;
	cout<<"Tail "<<tail->data <<endl;

	return 0;
}
```

(Q2) - Linked list implementation in java.

Code :

```
class LL {
	Node head;
	private int size;

	LL(){
		this.size = 0;
	}

	class Node{
		String data;
		Node next;

	Node(String data){
		this.data = data;
		this.next = null;
		size++;
		}
	}

	//add = first , last

	public void addFirst(String data){
		Node newNode = new Node(data);

		if(head == null)
		{
				head = newNode;
				return;
		}
		newNode.next = head;
		head = newNode;
	}
```

```
// add - last
public void addLast(String data){
	Node newNode = new Node(data);
	if(head == null){
		head = newNode;
		return;
	}

	Node curr = head;
	while(curr.next != null){
		curr = curr.next;
	}
	curr.next = newNode;
}


public void printList(){
	if(head == null){
		System.out.println("List is empty");
		return;
	}
	Node curr = head;
	while(curr != null){
		System.out.print(curr.data + " -> ");
		curr = curr.next;
	}
	System.out.println("NULL");

}

//delet first
public void deletFirst(){
	if(head == null){
		System.out.println("The list is empty");
		return;
	}
	size--;
	head = head.next;
}

//delet last
public void deletLast(){
```

```
if(head == null){
                System.out.println("The list is empty");
                return;
        }
size--;
if(head.next == null){
        head = null;
        return;
}
        Node secondLast = head;
        Node lastNode = head.next;
        while(lastNode.next != null){
                lastNode = lastNode.next;
                secondLast = secondLast.next;

        }
        secondLast.next = null;
}

public int getSize(){
return size;
}

public void reverseIterate(){
        if(head == null || head.next == null){
                return;
        }

        Node prevNode = head;
        Node currNode = head.next;

        while(currNode != null){
                Node nextNode = currNode.next;
                currNode.next = prevNode;

                //update
                prevNode = currNode;
                currNode = nextNode;
        }
        head.next = null;
        head = prevNode;
}

public Node reverseReurcive(Node head){
```

```
        if(head == null || head.next == null){
            return head;
        }
        Node newHead = reverseReurcive(head.next);
        head.next.next = head;
        head.next = null;

        return newHead;
    }

    public static void main(String args[]){
        LL list = new LL();
        list.addFirst("bhavya");
        list.printList();
        list.addFirst("is");
        list.printList();
        list.addLast("doing");
        list.printList();
        list.addLast("BTech ?");
        list.printList();

        list.deletFirst();
        list.printList();

        list.deletLast();
        list.printList();

        System.out.println(list.getSize());
        list.addFirst("Master");
        System.out.println(list.getSize());

        list.printList();
        list.reverseIterate();
        list.printList();

        list.printList();
        list.head = list.reverseReurcive(list.head);
        list.printList();
    }
}
```

(Q3) - Make a Linked List & add the following elements to it : (1, 5, 7, 3 , 8, 2, 3). Search for the number 7 & display its index.

Code :

```
public class Main {
    static class Node {
        public int data;
        public Node next;

        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static void main(String[] args) {
        Node head = new Node(1);
        Node tail = head;
        tail = insertAtTail(tail, 5);
        tail = insertAtTail(tail, 7);
        tail = insertAtTail(tail, 3);
        tail = insertAtTail(tail, 8);
        tail = insertAtTail(tail, 2);
        tail = insertAtTail(tail, 3);
        print(head);
        int index = findPosition(head, 7);
        if (index != -1)
        {
            System.out.println("\nThe index of 7 in the linked list is: " + (index+1));
        }
        else
        {
            System.out.println("7 is not found in the linked list.");
        }
    }
    public static Node insertAtTail(Node tail, int data) {
        Node temp = new Node(data);
        tail.next = temp;
        tail = temp;
        return tail;
    }
    public static void print(Node head) {
        Node temp = head;
        while (temp != null)
        {
            System.out.print(temp.data + " ");
```

```
            temp = temp.next;
        }
        System.out.println();
    }
    public static int findPosition(Node head, int num) {
        int count = 0;
        Node temp = head;
        while (temp != null && temp.data != num)
        {
            count++;
            temp = temp.next;
        }
        if (temp == null)
        {
            return -1;
        }
        else
        {
            return count;
        }
    }
}
```

Output :

```
1 5 7 3 8 2 3
The index of 7 in the linked list is: 3
```

(Q4) - Take elements(numbers in the range of 1-50) of a Linked List as input from the user. Delete all nodes which have values greater than 25

Code :

```
import java.util.*;
public class Main {
    static class Node {
        int value;
        Node next;

        Node(int value) {
            this.value = value;
            next = null;
        }
```

```
}

public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
   Node head = null;
   Node tail = null;

   System.out.println("Enter the elements of the linked list :");
   System.out.println("Enter any negative value to end input.");
   while (true) {
      int value = scanner.nextInt();
      if (value < 0)
      break;
      Node newNode = new Node(value);
      if (head == null)
      {
         head = newNode;
         tail = newNode;
      }
      else
      {
         tail.next = newNode;
         tail = newNode;
      }
   }
   System.out.println("Original Linked List:");
   printLinkedList(head);
   head = deleteNodesGreaterThan(head, 25);
   System.out.println("Linked List after deleting nodes greater than 25:");
   printLinkedList(head);
}
public static Node deleteNodesGreaterThan(Node head, int value) {
   if (head == null)
      return null;
   Node current = head;
   Node prev = null;
   while (current != null)
   {
      if (current.value > value)
      {
         if (prev == null)
         {
            head = current.next;
         }
```

```
            else
                prev.next = current.next;
        }
        else
        prev = current;
        current = current.next;
      }
      return head;
    }
    public static void printLinkedList(Node head) {
      Node current = head;
      while (current != null)
      {
        System.out.print(current.value + " ");
        current = current.next;
      }
      System.out.println();
    }
}
```

Output :

```
Enter the elements of the linked list :
Enter any negative value to end input.
2
3
5
9
6
25
65
10
23
64
29
31
10
7
9
-5
Original Linked List:
2 3 5 9 6 25 65 10 23 64 29 31 10 7 9
Linked List after deleting nodes greater than 25:
2 3 5 9 6 25 10 23 10 7 9
```