Name : Hardik Patel
Id : 202103032

1. Given two linked lists, insert nodes of the second list into the first list at alternate positions of first list. For example, if first list is 5->7->17->13->11 and second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and second list should become empty. The nodes of second list should only be inserted when there are positions available. For example, if the first list is 1->2->3 and second list is 4->5->6->7->8, then first list should become 1->4- >2->5->3->6 and second list to 7->8. Insertion must be done in place.

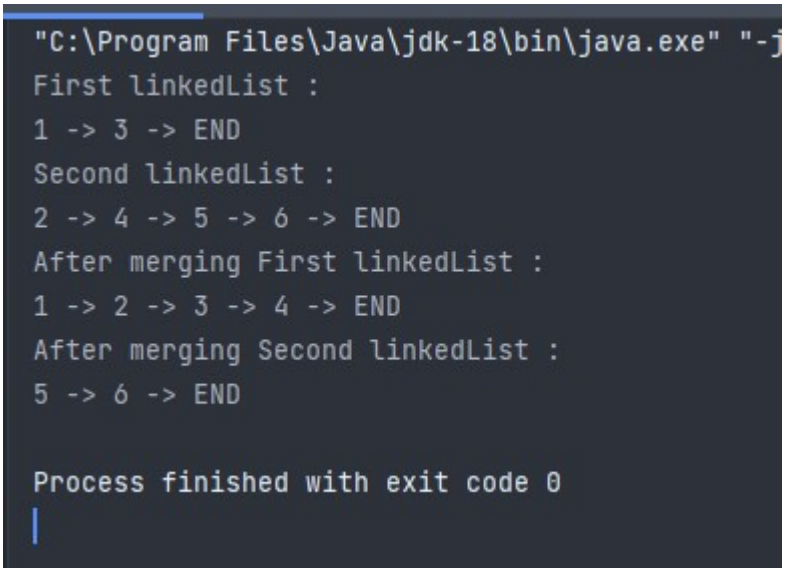
Code :

```
package com.DSA.LAB6;
public class AlternateLL {
    private Node head;
    private Node tail;
    public void insertFirst(int val) {
        //creating node
        Node node = new Node(val);
        node.next = head;
        head = node;
        //when list is empty
        if (tail == null) {
            tail = head;
        }
    }
    public void display() {
        //save head because head has to be at beginning
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.value + " -> ");
            temp = temp.next;
        }
        System.out.println("END");
    }
    // creating our own data type called Node
    private static class Node {
        private final int value;
        private Node next;
        public Node(int value) {
            this.value = value;
        }
    }
    //merge first and second list till one of the list's head reaches null
    void merge(AlternateLL list) {
        Node L1_head = head;
        Node L2_head = list.head;
        Node list1_next;
        Node list2_next;
        while (L1_head != null && L2_head != null) {
            // Save next pointers
            list1_next = L1_head.next;
            list2_next = L2_head.next;
            // insert
            L2_head.next = list1_next;
            L1_head.next = L2_head;
            // update current pointers for next iteration
            L1_head = list1_next;
```

```
            L2_head = list2_next;
        }
        list.head = L2_head;
    }
    public static void main(String[] args) {
        AlternateLL list1 = new AlternateLL();
        list1.insertFirst(5);
        list1.insertFirst(7);
        list1.insertFirst(17);
        list1.insertFirst(13);
        list1.insertFirst(11);
        System.out.println("First linkedList : ");
        list1.display();
        AlternateLL list2 = new AlternateLL();
        list2.insertFirst(12);
        list2.insertFirst(10);
        list2.insertFirst(2);
        list2.insertFirst(4);
        list2.insertFirst(6);
        System.out.println("Second linkedList : ");
        list2.display();
        list1.merge(list2);
        System.out.println("After merging First linkedList : ");
        list1.display();
        System.out.println("After merging Second linkedList : ");
        list2.display();
    }
}
```

Output :

```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-javaagent:C:\Program
First linkedList :
11 -> 13 -> 17 -> 7 -> 5 -> END
Second linkedList :
6 -> 4 -> 2 -> 10 -> 12 -> END
After merging First linkedList :
11 -> 6 -> 13 -> 4 -> 17 -> 2 -> 7 -> 10 -> 5 -> 12 -> END
After merging Second linkedList :
END

Process finished with exit code 0
```



2. Given a linked list, reverse every adjacent group of k nodes where k is a given positive integer. Input List: 1 —> 2 —> 3 —> 4 —> 5 —> 6 —> 7 —> 8 —> null For k = 3, Output: 3 —> 2 —> 1 —> 6 —> 5 —> 4 —> 8

Code:

```
package com.DSA.LAB6;
public class ReverseLL {
    private Node head;
    private Node tail;
    public void insertFirst(int val) {
        //creating node
        Node node = new Node(val);
        node.next = head;
        head = node;
        //when list is empty
        if (tail == null) {
            tail = head;
        }
    }
    public void display() {
        //save head because head has to be at beginning
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.value + " -> ");
            temp = temp.next;
        }
        System.out.println("END");
    }
    // creating our own data type called Node
    private static class Node {
        private final int value;
        private Node next;
        public Node(int value) {
            this.value = value;
        }
    }
```

```
    Node reverse(Node head, int k)
    {
        if(head == null)
            return null;
        Node current = head;
        Node next = null;
        Node prev = null;
        int count = 0;
        // Reverse first k nodes of linked list
        while (count < k && current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
            count++;
        }
        //next is now a pointer to (k+1)th node
        if (next != null)
            head.next = reverse(next, k);
        // prev = head
        return prev;
    }
    public static void main(String[] args) {
        ReverseLL list = new ReverseLL();
        list.insertFirst(9);
        list.insertFirst(8);
        list.insertFirst(7);
        list.insertFirst(6);
        list.insertFirst(5);
        list.insertFirst(4);
        list.insertFirst(3);
        list.insertFirst(2);
        list.insertFirst(1);
        System.out.println("Before Reverse : ");
        list.display();
        list.head = list.reverse(list.head, 3);
        System.out.println("After Reverse : ");
        list.display();
    }
}
```

Output :

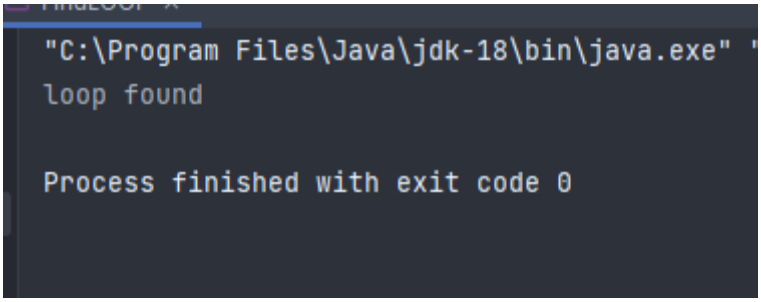
```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-javaagent:C:\
Before Reverse :
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> END
After Reverse :
3 -> 2 -> 1 -> 6 -> 5 -> 4 -> 9 -> 8 -> 7 -> END

Process finished with exit code 0
```

3. The networking routers are implemented using a linked list structure. The dynamic routing algorithm is implemented in the network. A loop in the network prevents information access to the other routers. Check if there is any closed loop that exists in the network or not. Assume that the address of the node can be treated as the network address

Code:

```
package com.DSA.LAB6;
class FindLOOP {
    Node head;
    int size;
    FindLOOP(){
        this.size = 0;
    }
    class Node {
        int data;
        Node next;
        Node(int data)
        {
            this.data = data;
            this.next = null;
        }
    }
    public void insertFirst(int data)
    {
        Node node = new Node(data);
        node.next = head;
        head = node;
        size+=1;
    }
    String findLoop()
    {
        Node oneStepPointer = head, twoStepPointer = head;
        int count = 0;
        while (oneStepPointer != null && twoStepPointer != null &&
twoStepPointer.next != null) {
            oneStepPointer = oneStepPointer.next;
            twoStepPointer = twoStepPointer.next.next;
            if (oneStepPointer == twoStepPointer) {
              return "loop found";
            }
        }
       return "Loop does not found";
    }
    public static void main(String[] args)
    {
        FindLOOP list = new FindLOOP();
        list.insertFirst(1);
        list.insertFirst(2);
        list.insertFirst(3);
        list.insertFirst(4);
        list.head.next.next.next.next = list.head;
        System.out.println( list.findLoop());
    }
}
```

Output :



4. A.Write a program that allots a vacant room to a new student of either MNC or ICT branch and can be put at the beginning or the end of the sequence, depending upon user input. Name, branch, and position (beginning/end) of student to be inserted should be user input. Keep the "overflow" condition. The program should also handle insertion in an empty list.

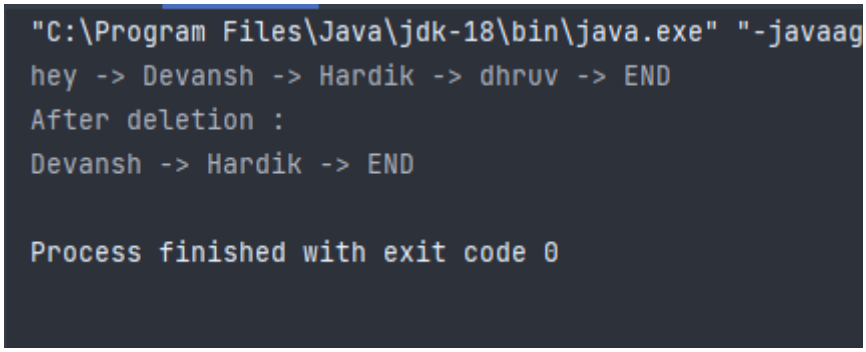
   B. Now if a student (ICT/MNC) graduates and vacates the room, write a program to update the list. Only the name of the student to be removed should be user input, not other information like branch or room number. Keep "underflow" condition. Assume that no two students have the same name. In both the programs, show results for inserting/deleting multiple entries

code:

```
package com.DSA.LAB6;
import com.Hardik.LL;
import java.util.Scanner;
public class TRY {
    static Scanner in = new Scanner(System.in);
    Node head;
    Node tail;
    int size;
    class Node{
        Node next;
        String data;
        Node(String data){
            this.data = data;
        }
    }
    public void insertFirst(String data) {
        Node node = new Node(data);
        node.next = head;
        head = node;
        if (tail == null) {
            tail = head;
        }
        size += 1;
    }
    public void insertLast(String data) {
        if (tail == null) {
            insertFirst(data);
            return;
        }
        Node node = new Node(data);
        tail.next = node;
```

```
        tail = node;
        size++;
    }
    public String deleteFirst() {
        String data = head.data;
        head = head.next;
        if (head == null) {
            tail = null;
        }
        size--;
        return data;
    }
    public void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("END");
    }
    public String  deleteLast() {
        if (size <= 1) {
            return deleteFirst();
        }
        Node secondLast = get(size - 2);
        String  val = tail.data;
        tail = secondLast;
        tail.next = null;
        size--;
        return val;
    }
    public Node get(int index) {
        Node node = head;
        for (int i = 0; i < index; i++) {
            node = node.next;
        }
        return node;
    }
    public static void main(String[] args) {
        TRY node = new TRY();
        node.insertFirst("Hardik");
        node.insertFirst("Devansh");
        node.insertLast("dhruv");
        node.insertFirst("hey");
        node.display();
        System.out.println("After deletion :");
        node.deleteFirst();
        node.deleteLast();
        node.display();
    }
}
```

output:


"C:\Program Files\Java\jdk-18\bin\java.exe" "-javaag
hey -> Devansh -> Hardik -> dhruv -> END
After deletion :
Devansh -> Hardik -> END
Process finished with exit code 0