# Lab 9 Linked List

public class L\_L

{

    Node head;

    private int size;

    //we are only calling constructor of Node when we are adding

    //new node otherwise we are only using reference variable therefore

    //we will get the size of list also while deleting a node

    //size will be decremented.

    L\_L()

    {

        this.size = 0;

    }

    class Node

    {

        String data;

        Node next;

        Node(String data)

        {

            this.data = data;

            this.next = null;

            size++;

        }

    }

    //add first, add last

    public void addFirst(String data)

    {

        Node newNode = new Node(data);

        if(head == null)

        {

            head = newNode;

            return;

        }

        newNode.next = head;

        head = newNode;

    }

    //Adding element to the last node

    public void addLast(String data)

    {

        Node newNode = new Node(data);

        if(head == null)

        {

            head = newNode;

            return;

        }

        Node currNode = head;

        while(currNode.next != null)

        {

            currNode = currNode.next;

        }

        currNode.next = newNode;

    }

    //Deleting first element

    public void delFirst()

    {

        if(head == null)

        {

            System.out.println("List is empty");

            return;

        }

        size--;

        head = head.next;

    }

    //Deleting last element

    public void delLast()

    {

        if(head == null)

        {

            System.out.println("List is empty");

            return;

        }

        size--;

        if(head.next == null)

        {

            head = null;

            return;

        }

        Node currNode = head;

        Node secLastNode=currNode;

        while(currNode.next != null) //corner case: if there is only one

        {                            //node in the linked list

            secLastNode = currNode;

            currNode = currNode.next;

        }

        secLastNode.next = null;

    }

    //Print

    public void printList()

    {

        if(head == null)

        {

            System.out.println("List is empty");

            return;

        }

        Node currNode = head;

        while(currNode != null)

        {

            System.out.print(currNode.data + " -> ");

            currNode = currNode.next;

        }

        System.out.println("NULL");

    }

    //Returning Size

    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;

            //nextNode = nextNode.next;

        }

        head.next = null;

        head = prevNode;

    }

    //Reversing the LinkedList

    public Node reverseRecursive(Node head)

    {

        if(head == null || head.next == null)

        {

            return head;

        }

        Node newHead = reverseRecursive(head.next);

        head.next.next = head;

        head.next = null;

        return newHead;

    }

    public static void main(String[] args) {

        L\_L list = new L\_L();

        list.addFirst("a");

        list.addFirst("is");

        list.addFirst("This");

        list.addLast("List");

        list.addLast("of");

        list.addLast("Students");

        list.addFirst("And");

        list.printList();

        list.delLast();

        list.delLast();

        list.delFirst();

        list.printList();

        System.out.println(list.getSize());

        //list.reverseIterate();

        //list.printList();

        list.head = list.reverseRecursive(list.head);

        list.printList();

    }

}