**C-3.25**

Algorithm for concatenation

Let us assume that the two linked lists are referenced by **head1** and **head2** .

1. If the first linked list is empty then return **head2**.

2. If the second linked list is empty then return **head1**.

3. Store the address of the starting node of the first linked  
list in a pointer variable, say **next**.

4. Move the **next** to the last node of the linked list through simple linked list traversal  
technique.

5. Store the address of the first node of the second linked list in the next field of the node pointed by **next**. Return **head1.**

**C-3.12**

*/\*\*  
 \* Created by op on 3/25/2021.  
 \*/***public class SinglyLinkedList <E> {  
 private Node <E> head=null;  
 private Node <E> tail=null;  
 int size=0;  
 public boolean isEmpty()  
 {  
 return size==0;  
 }  
 public int size()  
 {  
 return size;  
 }  
 public E First()  
 {  
 if (isEmpty()) return null;  
 return head.getElement();  
 }  
 public E Last()  
 {  
 if (isEmpty()) return null;  
 return tail.getElement();  
 }  
 public void addFirst(E element)  
 {  
 head=new Node<E>(element,head);  
 if (size==0)  
 head=tail;  
 size++;  
 }  
 public void addLast(E element)  
 {  
 Node<E> newest=new Node<E>(element,tail);  
 if (size==0)  
 head=newest;  
 else  
 tail.setNext(newest);  
 tail=newest;  
 size++;  
 }  
 public E removedFirst()  
 {  
 if (isEmpty()) return null;  
 E deleted = head.getElement();  
 head = head.getNext();  
 size--;  
 if (size == 0)  
 tail = null;  
 return deleted;  
 }  
 public void reverse()  
 {  
 Node reversedPart = null;  
 Node current = head;  
  
 while(current != null)  
 {  
 Node next = current.next;  
 current.next = reversedPart;  
 reversedPart = current;  
 current = next;  
 }  
 head = reversedPart;  
 }  
 public void rotateLeft()  
 {  
 Node temp = head;  
 if (head != null) {  
  
 if (head.getNext() != null) {  
 head = head.getNext();  
 }  
 }  
  
 Node tail;  
 if (head.getNext() != null) {  
  
 tail = head.getNext();  
 } else {  
  
 tail = head;  
 }  
  
 while(tail.getNext() != null)  
 {  
 if (tail.getNext() != null) {  
 tail = tail.getNext();  
 }  
  
 }  
 tail.setNext(temp);  
 temp.setNext(null);  
 }  
 public String print()  
 {  
 Node<E> i=head;  
 String all="";  
 while (i!=null)  
 {  
 all=all+i.getElement().toString()+"\n";** i=i.getNext();  
  
 }  
 **return** all;  
 }  
  
 **private static class** Node<E>  
 {  
 **private** E **element**;  
 **private** Node<E> **next**;  
  
 **public** Node(E element, Node<E> next) {  
 **this**.**element** = element;  
 **this**.**next** = next;  
 }  
  
 **public** E getElement() {  
 **return element**;  
 }  
  
 **public void** setElement(E element) {  
 **this**.**element** = element;  
 }  
  
 **public** Node<E> getNext() {  
 **return next**;  
 }  
  
 **public void** setNext(Node<E> next) {  
 **this**.**next** = next;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Node{"** +  
 **"element="** + **element** +  
 **", next="** + **next** +  
 **'}'**;  
 }  
 }  
  
  
  
}

*/\*\*  
 \* Created by op on 3/25/2021.  
 \*/***public class TestSinglyLinkedList {  
 public static void main(String[] args) {  
 SinglyLinkedList <Integer> list=new SinglyLinkedList<>();  
 SinglyLinkedList <Integer> list2=new SinglyLinkedList<>();  
 list.addFirst(1);  
 list.addFirst(2);  
 list.addFirst(3);  
 list.addFirst(4);  
 list.addFirst(5);  
  
 int n=list.size();  
 for (int i = 0; i < n; i++) {  
*// System.out.println(list.removedFirst());  
// System.out.println(list.size());  
// System.out.println(list2.size());* System.*out*.println(list.print());  
  
  
 }  
 list.reverse();  
 list.rotateLeft();  
 System.*out*.println(list.print());  
  
  
  
 }  
}**

**C3.28**

**public void reverse()  
 {  
 Node reversedPart = null;  
 Node current = head;  
  
 while(current != null)  
 {  
 Node next = current.next;  
 current.next = reversedPart;  
 reversedPart = current;  
 current = next;  
 }  
 head = reversedPart;  
 }**