Display Reverse (recursive) - Linked List

Try First, Check Solution later

1. You should first read the question and watch the question video.  
2. Think of a solution approach, then try and submit the question on editor tab.  
3. We strongly advise you to watch the solution video for prescribed approach.

Question

1. You are given a partially written LinkedList class.  
2. You are required to complete the body of displayReverse and displayReverseHelper functions. The function are expected to print in reverse the linked list without actually reversing it.   
3. Input and Output is managed for you.   
  
Note -> The online judge can't force you to write recursive function. But that is what the expectation is, the intention in to help you learn.

Input Format

Input is managed for you

Output Format

Output is managed for you

Constraints

1. Time complexity -> O(n)  
2. Space complexity -> O(n)

Sample Input

11  
1 2 3 4 5 6 7 8 9 10 11  
100  
200

Sample Output

1 2 3 4 5 6 7 8 9 10 11   
11 10 9 8 7 6 5 4 3 2 1   
200 1 2 3 4 5 6 7 8 9 10 11 100

import java.io.\*;

import java.util.\*;

public class Main {

public static class Node {

int data;

Node next;

}

public static class LinkedList {

Node head;

Node tail;

int size;

void addLast(int val) {

Node temp = new Node();

temp.data = val;

temp.next = null;

if (size == 0) {

head = tail = temp;

} else {

tail.next = temp;

tail = temp;

}

size++;

}

public int size() {

return size;

}

public void display() {

for (Node temp = head; temp != null; temp = temp.next) {

System.out.print(temp.data + " ");

}

System.out.println();

}

public void removeFirst() {

if (size == 0) {

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

} else if (size == 1) {

head = tail = null;

size = 0;

} else {

head = head.next;

size--;

}

}

public int getFirst() {

if (size == 0) {

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

return -1;

} else {

return head.data;

}

}

public int getLast() {

if (size == 0) {

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

return -1;

} else {

return tail.data;

}

}

public int getAt(int idx) {

if (size == 0) {

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

return -1;

} else if (idx < 0 || idx >= size) {

System.out.println("Invalid arguments");

return -1;

} else {

Node temp = head;

for (int i = 0; i < idx; i++) {

temp = temp.next;

}

return temp.data;

}

}

public void addFirst(int val) {

Node temp = new Node();

temp.data = val;

temp.next = head;

head = temp;

if (size == 0) {

tail = temp;

}

size++;

}

public void addAt(int idx, int val) {

if (idx < 0 || idx > size) {

System.out.println("Invalid arguments");

} else if (idx == 0) {

addFirst(val);

} else if (idx == size) {

addLast(val);

} else {

Node node = new Node();

node.data = val;

Node temp = head;

for (int i = 0; i < idx - 1; i++) {

temp = temp.next;

}

node.next = temp.next;

temp.next = node;

size++;

}

}

public void removeLast() {

if (size == 0) {

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

} else if (size == 1) {

head = tail = null;

size = 0;

} else {

Node temp = head;

for (int i = 0; i < size - 2; i++) {

temp = temp.next;

}

tail = temp;

tail.next = null;

size--;

}

}

public void removeAt(int idx) {

if (idx < 0 || idx >= size) {

System.out.println("Invalid arguments");

} else if (idx == 0) {

removeFirst();

} else if (idx == size - 1) {

removeLast();

} else {

Node temp = head;

for (int i = 0; i < idx - 1; i++) {

temp = temp.next;

}

temp.next = temp.next.next;

size--;

}

}

private Node getNodeAt(int idx) {

Node temp = head;

for (int i = 0; i < idx; i++) {

temp = temp.next;

}

return temp;

}

public void reverseDI() {

int li = 0;

int ri = size - 1;

while (li < ri) {

Node left = getNodeAt(li);

Node right = getNodeAt(ri);

int temp = left.data;

left.data = right.data;

right.data = temp;

li++;

ri--;

}

}

public void reversePI() {

if (size <= 1) {

return;

}

Node prev = null;

Node curr = head;

while (curr != null) {

Node next = curr.next;

curr.next = prev;

prev = curr;

curr = next;

}

Node temp = head;

head = tail;

tail = temp;

}

public int kthFromLast(int k) {

Node slow = head;

Node fast = head;

for (int i = 0; i < k; i++) {

fast = fast.next;

}

while (fast != tail) {

slow = slow.next;

fast = fast.next;

}

return slow.data;

}

public int mid() {

Node f = head;

Node s = head;

while (f.next != null && f.next.next != null) {

f = f.next.next;

s = s.next;

}

return s.data;

}

public static LinkedList mergeTwoSortedLists(LinkedList l1, LinkedList l2) {

LinkedList ml = new LinkedList();

Node one = l1.head;

Node two = l2.head;

while (one != null && two != null) {

if (one.data < two.data) {

ml.addLast(one.data);

one = one.next;

} else {

ml.addLast(two.data);

two = two.next;

}

}

while (one != null) {

ml.addLast(one.data);

one = one.next;

}

while (two != null) {

ml.addLast(two.data);

two = two.next;

}

return ml;

}

public static Node midNode(Node head, Node tail) {

Node f = head;

Node s = head;

while (f != tail && f.next != tail) {

f = f.next.next;

s = s.next;

}

return s;

}

public static LinkedList mergeSort(Node head, Node tail) {

if (head == tail) {

LinkedList br = new LinkedList();

br.addLast(head.data);

return br;

}

Node mid = midNode(head, tail);

LinkedList fsh = mergeSort(head, mid);

LinkedList ssh = mergeSort(mid.next, tail);

LinkedList sl = mergeTwoSortedLists(fsh, ssh);

return sl;

}

public void removeDuplicates() {

LinkedList res = new LinkedList();

while (this.size() > 0) {

int val = this.getFirst();

this.removeFirst();

if (res.size() == 0 || val != res.tail.data) {

res.addLast(val);

}

}

this.head = res.head;

this.tail = res.tail;

this.size = res.size;

}

public void oddEven() {

LinkedList odd = new LinkedList();

LinkedList even = new LinkedList();

while (this.size > 0) {

int val = this.getFirst();

this.removeFirst();

if (val % 2 == 0) {

even.addLast(val);

} else {

odd.addLast(val);

}

}

if (odd.size > 0 && even.size > 0) {

odd.tail.next = even.head;

this.head = odd.head;

this.tail = even.tail;

this.size = odd.size + even.size;

} else if (odd.size > 0) {

this.head = odd.head;

this.tail = odd.tail;

this.size = odd.size;

} else if (even.size > 0) {

this.head = even.head;

this.tail = even.tail;

this.size = even.size;

}

}

public void kReverse(int k) {

LinkedList prev = null;

while (this.size > 0) {

LinkedList curr = new LinkedList();

if (this.size >= k) {

for (int i = 0; i < k; i++) {

int val = this.getFirst();

this.removeFirst();

curr.addFirst(val);

}

} else {

int sz = this.size;

for (int i = 0; i < sz; i++) {

int val = this.getFirst();

this.removeFirst();

curr.addLast(val);

}

}

if (prev == null) {

prev = curr;

} else {

prev.tail.next = curr.head;

prev.tail = curr.tail;

prev.size += curr.size;

}

}

this.head = prev.head;

this.tail = prev.tail;

this.size = prev.size;

}

private void displayReverseHelper(Node node){

// write your code here

if(node == null){

return;

}

displayReverseHelper(node.next);

System.out.print(node.data + " ");

}

public void displayReverse(){

displayReverseHelper(head);

System.out.println();

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n1 = Integer.parseInt(br.readLine());

LinkedList l1 = new LinkedList();

String[] values1 = br.readLine().split(" ");

for (int i = 0; i < n1; i++) {

int d = Integer.parseInt(values1[i]);

l1.addLast(d);

}

int a = Integer.parseInt(br.readLine());

int b = Integer.parseInt(br.readLine());

l1.display();

l1.displayReverse();

l1.addLast(a);

l1.addFirst(b);

l1.display();

}

}