*“Don’t Let Yesterday Take Up Too Much Of Today.” – Will Rogers*

Dear reader, are you enjoying doing coding or not? Nevertheless, I welcome you to the new problem: ‘[***Odd Even Linked List***](https://www.pepcoding.com/resources/online-java-foundation/linked-lists/odd-even-linked-list-official/ojquestion)***’***.

***Problem Statement***

* You are given a partially written LinkedList class.
* You are required to complete the body of oddEven function. (Input and Output is managed for you.)
* The function is expected to tweak the list such that all odd values are followed by all even values.
* The relative order of elements should not change. Also, take care of the cases when there are no odd or no even elements.
* Make sure to properly set head, tail and size as the function will be tested by calling addFirst and addLast.

To understand the problem statement if you are not able to get it, I recommend you to watch the [question video](https://www.youtube.com/watch?v=58o1--_XGC4) associated with this problem.

***Example***

* Consider, you are given the following linked list: *2 -> 3 -> 8 -> 1 -> 5 -> 4 -> 9 -> null*. Then the resultant linked list should be : *3 -> 1 -> 5 -> 9 -> 2 -> 8 -> 4 -> null*.
* If there are no odd elements like for the linked list: 2 -> 4 -> 8 -> 6 -> null, then the resultant linked list shall remain the same.
* Similarly, if there are no even elements, for eg in a list like: 5 -> 3 -> 7 -> null, then the resultant linked list shall remain the same in this case also.

***Deducing Algorithm***

Let us maintain two linked lists, which are empty initially, named *odd* and *even*. At the end of the algorithm, odd should contain all the odd-valued elements, whereas the even should contain all the even-valued elements (both should be in the same order in which they appear in the original linked list).

Similar to what we did for [*Remove Duplicates in Sorted List*](https://www.pepcoding.com/resources/online-java-foundation/linked-lists/remove-duplicates-official/ojquestion), we will keep on removing the first node of the original linked list using ***removeFirst()*** method.

Now, if the element removed is odd-valued, then we will add a node with the same value at the end of *odd* using ***addLast*()** method.

Else, the element is even-valued, hence we will add a node with the same value at the end of *even* using ***addLast()*** method.

Finally, all the elements from the original linked list will be removed and will either be placed in the *odd* or *even* linked list.

**Q)** Are we preserving the original order among the odd elements and among the even elements?

**R)** **Yes**. Since, we are removing nodes one by one from the original linked list in order and placing them in the last of odd/even linked lists in the same order, hence we are able to preserve the relative ordering.

Now, there are three cases:

1. If both odd and even linked lists are non-empty (head != null):
   * Then we will link the next pointer of odd’s last node to the head node of the even linked list.
   * Now, we will make the head of ***this*** linked list as the odd’s head, and ***this*** tail as the even’s tail.
   * Also, we will make the size equal to the sum of sizes of even and odd linked lists.
2. If there are no odd elements:

* We can simply make our linked list’s head as odd’s head and tail as odd’s tail and size equal to the size of the odd linked list.

1. If there are no even elements:

* We can simply make our linked list’s head as even’s head and tail as even’s tail and size equal to the size of the even linked list.

Please refer to the [solution video](https://www.youtube.com/watch?v=calrG6RJ9qI) if you find difficulty in understanding the algorithm completely.

***Suggestion***: Don’t worry about how head and tail pointers will manage themselves in ***addLast()***, ***removeFirst()*** functions. They were written by you only in previous problems, and now you are using them to solve more difficult problems. Clustering your thoughts around these functions can result in you failing in not coming up with an algorithm for the current problem statement.

***Implementation (Java)***

Please I request you to give it a try before reading the code! Don’t just blindly copy the code and submit it on any portal for the sake of completion. Doing so will not help you build an insight about Data Structures & Algorithms.

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 void addFirst(int val) {

Node temp = new Node();

temp.data = val;

temp.next = head;

head = temp;

if (size == 0) {

tail = temp;

}

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--;

}

}

private Node getNodeAt(int idx) {

Node temp = head;

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

temp = temp.next;

}

return temp;

}

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 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.oddEven();

l1.display();

l1.addFirst(a);

l1.addLast(b);

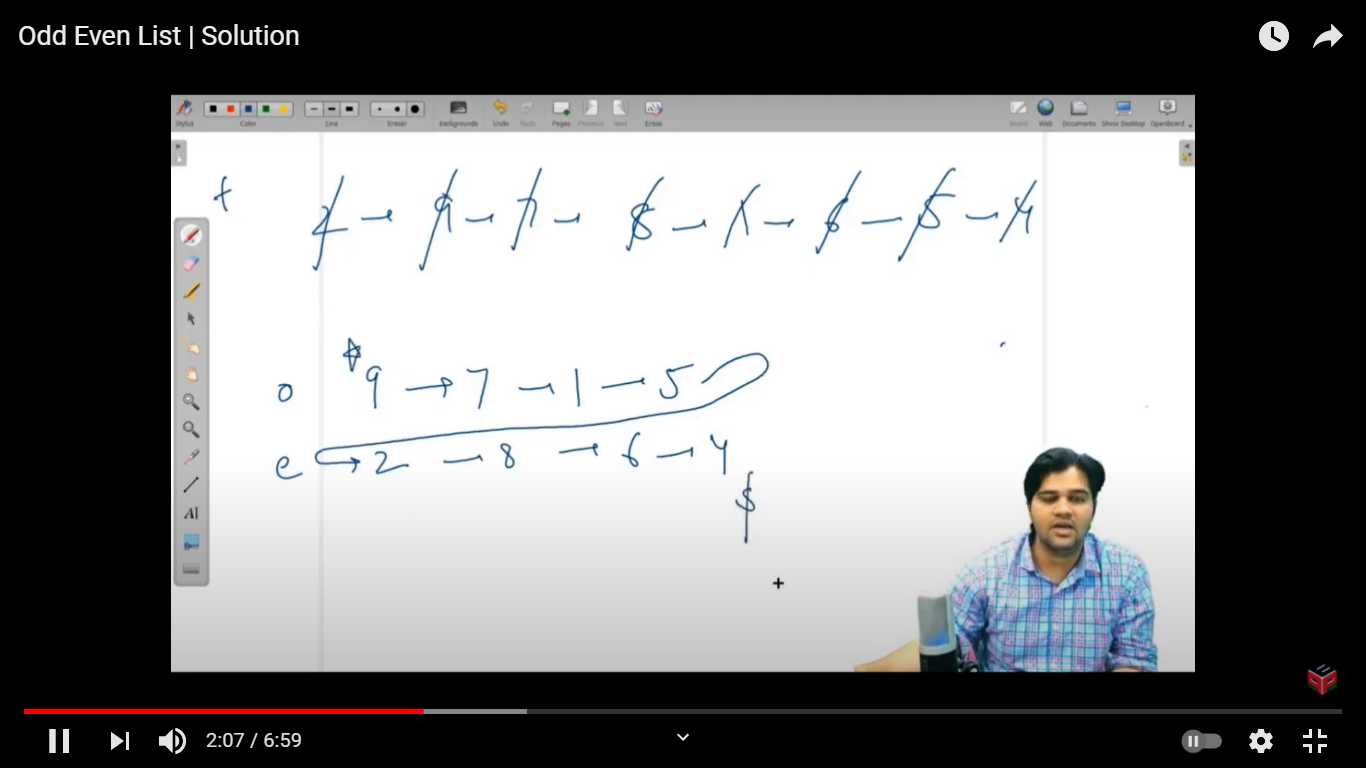
l1.display();

}

}

This code is written and explained by our team in [this video](https://www.youtube.com/watch?v=calrG6RJ9qI). Please refer to it if you are stuck somewhere.

You should perform a **dry run** of the algorithm on some examples, draw the recursion tree, to get a better understanding. It is also explained in the [same video](https://www.youtube.com/watch?v=calrG6RJ9qI).



***Time & Space Complexity Analysis***

**Time Complexity** - We are removing all nodes of original linked list one-by-one, which takes n \* O(1) = O(n) time. Then, we are either adding them to odd linked list or the even linked list, which will again take n \* O(1) = O(n) time.

Hence, the total time complexity = O(n + n) = O(2 \* n) = ***O(n)*** only.

**Space Complexity** - Similar to what we discussed in the [previous problem,](https://www.pepcoding.com/resources/online-java-foundation/linked-lists/kreverse-linkedlist-official/ojquestion) before adding a new node, we are ***REMOVING*** (deallocating space) a node from the original list, hence we are taking constant O(1) space.

I hope you enjoyed solving the problem with me. I will bring another problem for you to solve, which is named ‘[***K Reverse in Linked List***](https://www.pepcoding.com/resources/online-java-foundation/linked-lists/kreverse-linkedlist-official/ojquestion)’. *Happy Coding*!

Contributor : [Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)