*“A successful man is one who can lay a firm foundation with the bricks others have thrown at him.” - David Brinkley*

Dear reader, I hope you enjoy solving these problems. We are going to solve a very popular Linked List problem, and the problem’s name is “[***K Reverse in Linked List***](https://www.pepcoding.com/resources/online-java-foundation/linked-lists/kreverse-linkedlist-official/ojquestion)”.

I can place a bet that this question can haunt a lot of programmers, if they don’t know the basic functions like ***addFirst(), addLast(), removeFirst(), removeLast()***.

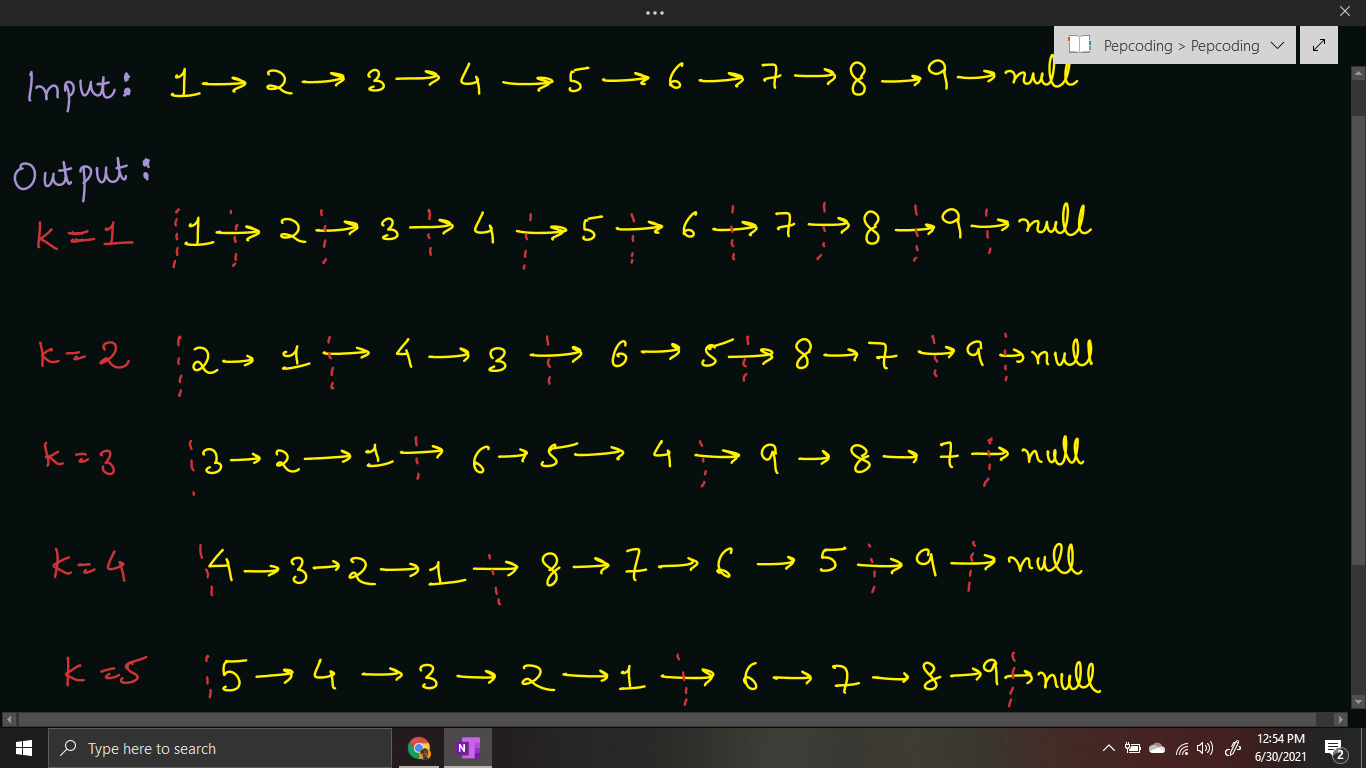
But since you are following this topic with us from the right basics, I can assure you that solving this problem will be just like any other day of coding . Alright, enough talk, let’s jump onto the problem statement!

***Problem Statement***

* You are given a partially written LinkedList class. (Input and Output is managed for you.)
* You are required to complete the body of the ***kReverse*** function.
* The function is expected to tweak the list such that all groups of k elements in the list get reversed and linked.
* If the last set has less than k elements, leave it as it is (***don't reverse***).

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***



***Deducing Algorithm***

Before solving this problem, I request you to solve some easier problems based on this concept. We have covered problems: ‘[***Odd Even Linked List***](https://www.pepcoding.com/resources/online-java-foundation/linked-lists/odd-even-linked-list-official/ojquestion)’ and ‘[***Remove Duplicates from Sorted Linked List***](https://www.pepcoding.com/resources/online-java-foundation/linked-lists/remove-duplicates-official/ojquestion)’, and this problem follows a similar approach to those discussed there.

* We will maintain two linked lists: ***previous*** and ***current*** which will be empty initially. Now, we will keep on removing the first element of the original linked list one by one using ***removeFirst()*** method. Let the current element removed be equal to *val*.
* If the size of the *current* linked list is less than k, then we can add a node in the front of the current linked list (with value = val), using the ***addFirst()*** method.
* By doing addFirst() one by one, we are adding the elements in ***reverse order***, hence the segment of nodes in the current linked list will be reversed from the order in the original linked list.

*Note*: addFirst() will manage the corner cases if the current linked list is empty or not. Do not worry about these functions.

* Else, the current linked list has size equal to k. Hence we got a segment of k nodes, which are in reverse order in the *current* linked list. We will place this whole segment at the last of the *previous* linked list.
* By placing the whole segment in last, I mean that, we will make the next pointer of the tail node of the previous linked list point to the head of the current linked list. Also, update the tail of the previous linked list as the tail of the current linked list, and increase previous’ size by current’s size.
* We will have to handle a corner case in this case, when the previous linked list is empty. In this case, tail will point to null, hence updating the next pointer of null node will give a run-time error.
* Thus, before appending the current linked list to the previous list, we will check if the previous list is empty or not (using previous’ head != null). If it is empty, then we will simply make previous = current.
* Also, we will make the current list empty by making head and tail point to null, so that we can process the remaining original linked list.

**Q)** Have we handled all the corner cases? What about the condition when the last segment has nodes less than k?

**R)** So, it seems that the question was not that easy! Before removing the element from the original linked list, we will have to check if the size of the remaining list is less than k, then we need not reverse the linked list, hence we can directly append the remaining nodes to the *previous* list by *removingFirst()* in original linked list and *addLast()* in previous linked list.

* After completing the above steps, we will be left with an empty original list, and the previous list will contain all the nodes in k-reversed form. Hence, we will update *this.head = prev.head, this.tail = prev.tail* and *this.size = prev.size*.

Please refer to the [solution video](https://www.youtube.com/watch?v=jhm2pYGFIos&list=TLGGZel-7zJiIk0zMDA2MjAyMQ) if you find difficulty in understanding the algorithm completely.

***Implementation (Java)***

Please I request you to give it a try before reading the code!

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 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;

}

}

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 k = Integer.parseInt(br.readLine());

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

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

l1.display();

l1.kReverse(k);

l1.display();

l1.addFirst(a);

l1.addLast(b);

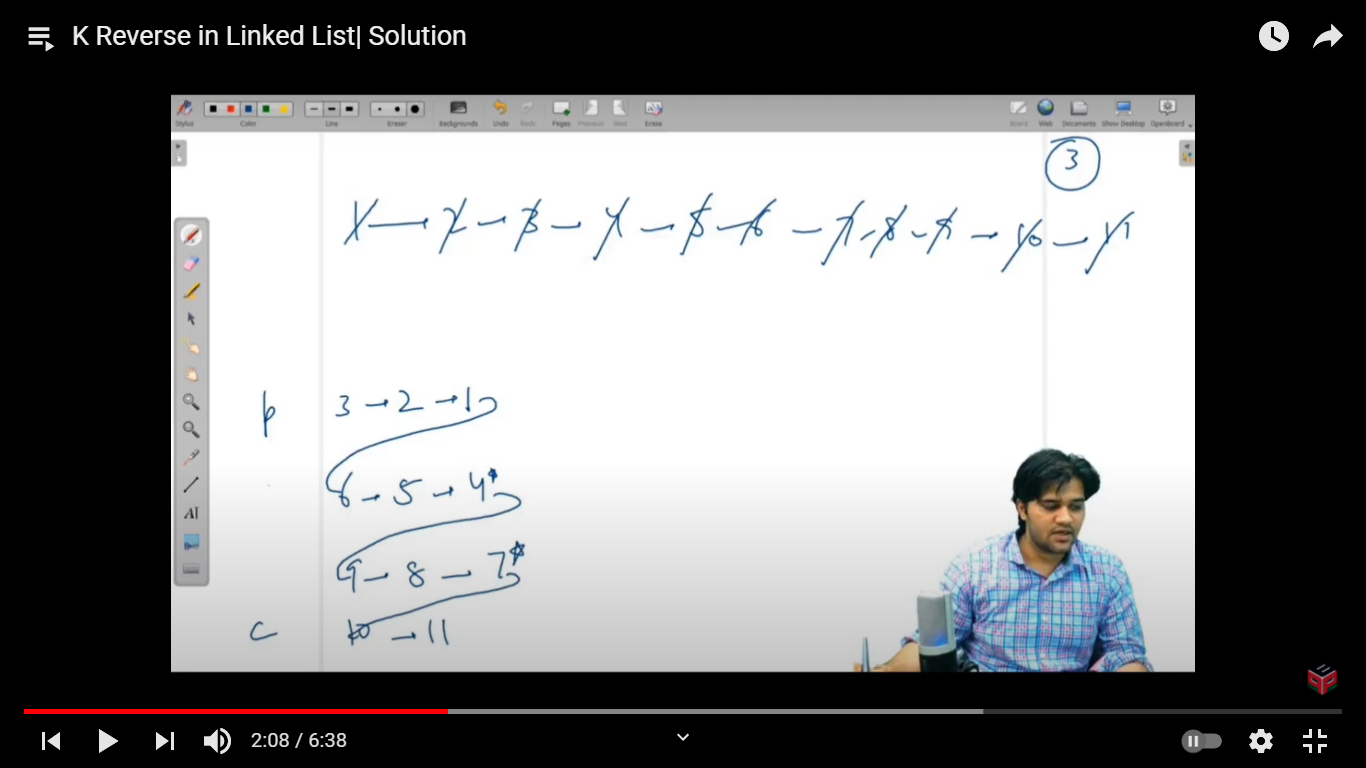
l1.display();

}

}

This code is written and explained by our team in [this video](https://www.youtube.com/watch?v=jhm2pYGFIos&list=TLGGZel-7zJiIk0zMDA2MjAyMQ) from *[2:25, 6:35]*. Please refer to it if you are stuck somewhere.

You should perform a **dry run** of the algorithm on some examples, to get a better understanding. It is also explained in the [same video](https://www.youtube.com/watch?v=jhm2pYGFIos&list=TLGGZel-7zJiIk0zMDA2MjAyMQ).



***Time & Space Complexity Analysis***

**Time Complexity** - We are processing each node from the original linked list, and adding it in the current linked list (or directly to the previous linked list if size < k).

Hence, time complexity is equal to ***O(n)*** where n = number of nodes in linked list.

**Space Complexity** - We are not using any extra space in this solution. As discussed in previous problems, we are ***REMOVING*** nodes from the given list and then only adding them into the current (or previous) list. Hence, only constant space O(1) is taken, to form new linked list’s head, tail & size integer variables.

***Extra Gyaan (Knowledge)***

**Q)** What if the last part also needs to be reversed irrespective of size, i.e. if there are less than k nodes in the remaining list, then also reverse it?

**R)** This will make the problem more simple. We will not have to check at each node whether the number of remaining nodes is less than k or not. We can simply keep on doing *addFirst()* on the current linked list, and we will get to the answer.

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

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