**LAB 4 REPORT**

1/LinkListApp.java

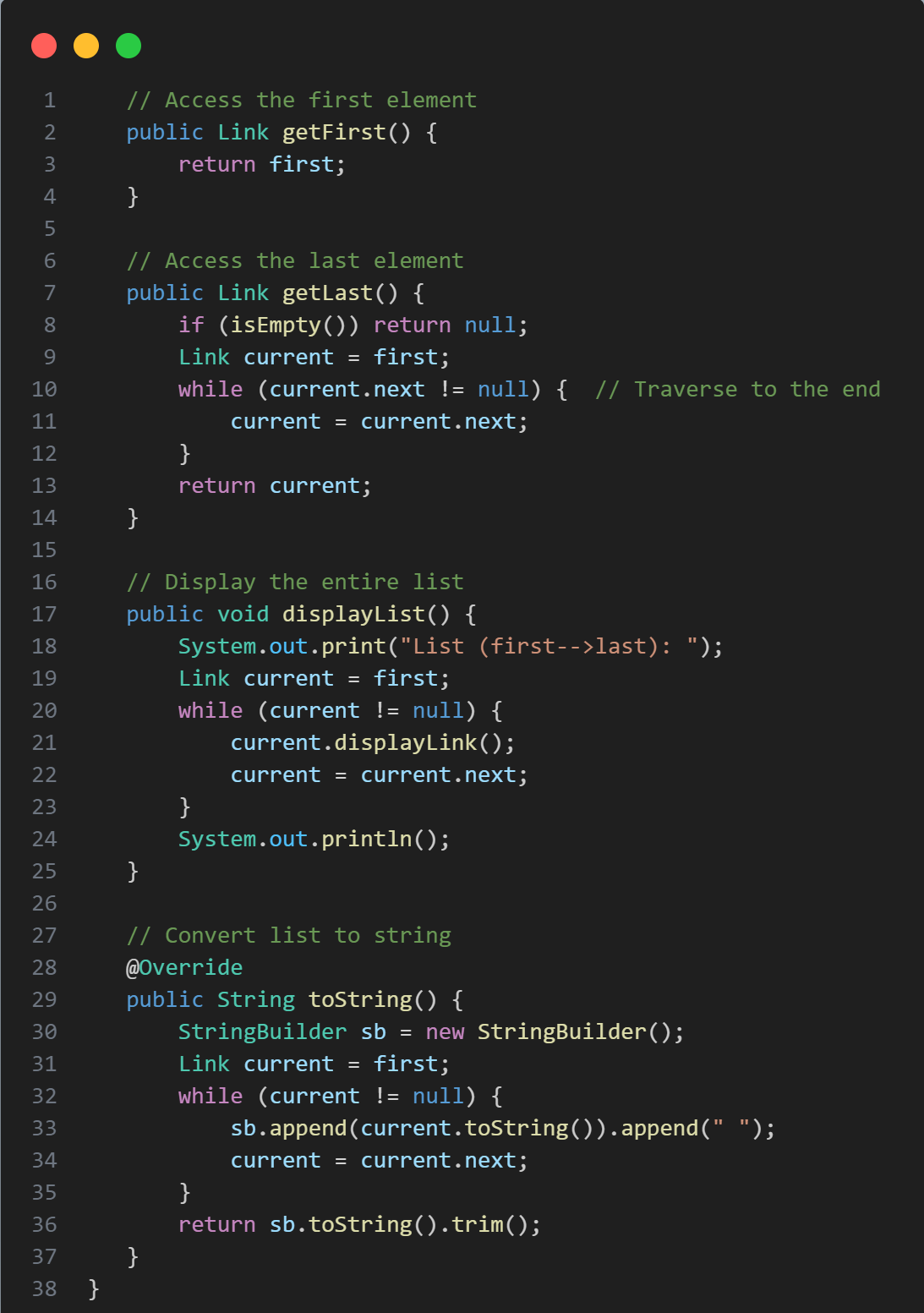
Initial output of LinkListApp.java

A screen shot of a computer

Description automatically generated

Add methods to access the first and the last element of the list, add toString() methods:

* getFirst(): returns first, which is a reference to the first Link in the list. Since first is always pointing to the start of the list, this allows quick access to that element.
* getLast(): starts from first and iterates through each Link using a while loop, moving to the next link (current.next) until it reaches the end of the list (where current.next is null).
* toString() in Link: Formats a single link as {iData, dData}, making individual links easier to print.
* toString() in LinkList: Iterates through the entire list, building a string representation for simple, readable list output.



Output after add methods:

A screen shot of a computer

Description automatically generated

2/ LinkStackApp.java

In this problem, we write an application to reverse a list using a stack

The reverseList method:

* Accepts a list (inputList) as input.
* Uses a stack (Stack<Integer>) to hold the elements of the list in reverse order.
* The elements are pushed onto the stack, and then popped off and added to a new list (reversedList), resulting in the reversed order.

A screen shot of a computer program

Description automatically generated

Test:

A screen shot of a computer program

Description automatically generated

Output:

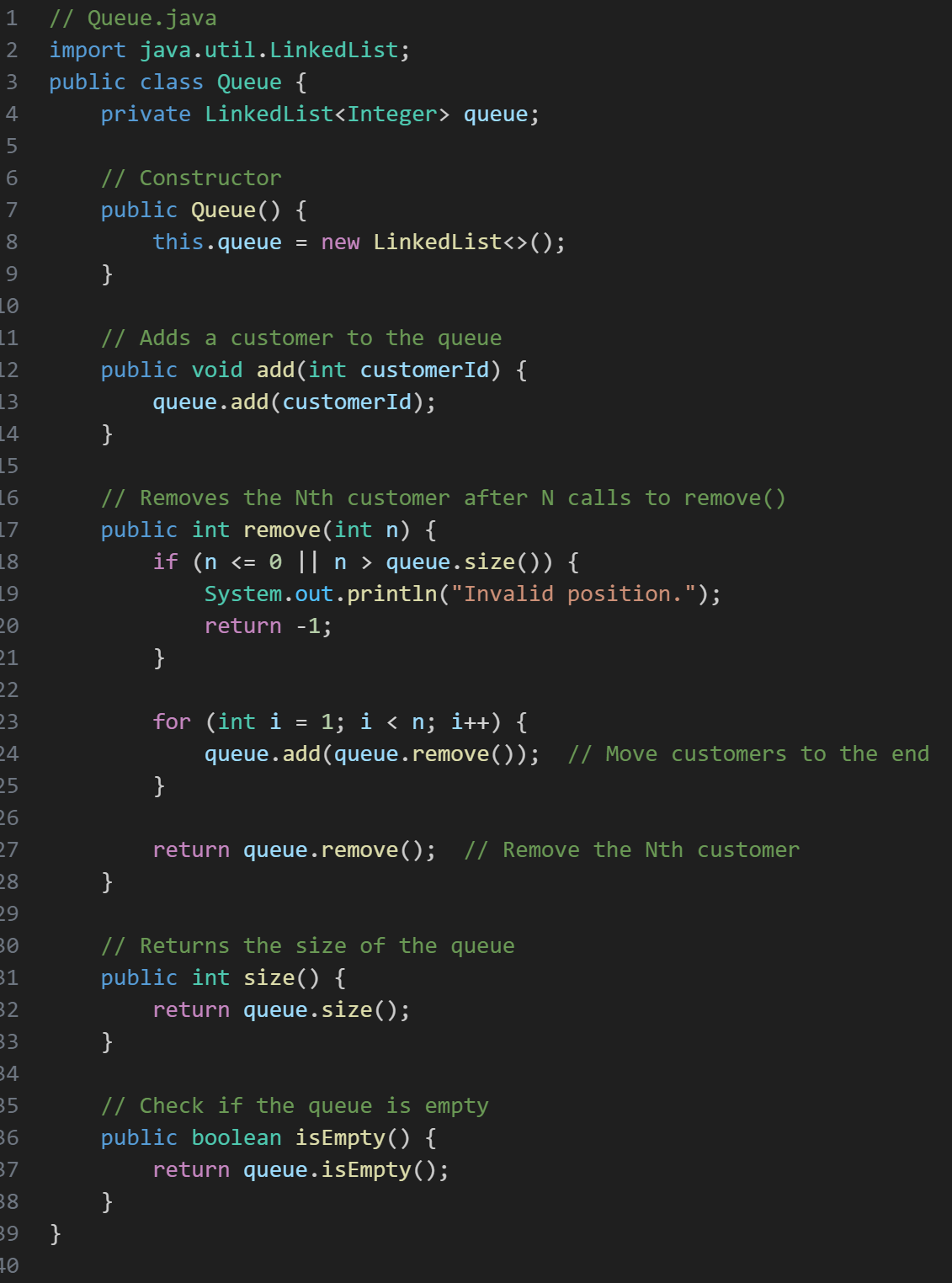


3/ LinkQueueApp.java

In this problem, we separate the solution into 3 class,

1. Queue.java:

* Simulates the queue of customers using a linked list structure and support the custom remove() and size() methods.
* The Queue class implements a queue using a LinkedList.
* It has methods to add customers, remove customers based on an Nth call to remove(), and get the size of the queue. The remove() method moves customers around to ensure that the Nth customer is removed after N calls.



1. Customer.java:

* Represents a customer who is served for a random amount of time (between 1 and 10 seconds) when they are created. The serve() method simulates the serving time by sleeping for that duration.

A screen shot of a computer program

Description automatically generated

1. LinkQueueApp.java:

* Runs the simulation, adds customers to the queue, and removes them based on the remove() method logic.
* This is the main application. It adds 10 customers to the queue, then simulates serving them by randomly removing and serving customers at different times.
* The rate at which customers join the queue and the time they are served will affect the simulation’s behavior.

A computer screen shot of a program code

Description automatically generated

Output:

A screenshot of a computer

Description automatically generated

4/ Josephus Problem

CircularLinkedList Class:

* This class uses a circular linked list where each Node holds the person’s number.
* The addPerson() method creates the circular linked list by linking the last person to the first.

Recursive Elimination (Josh method):

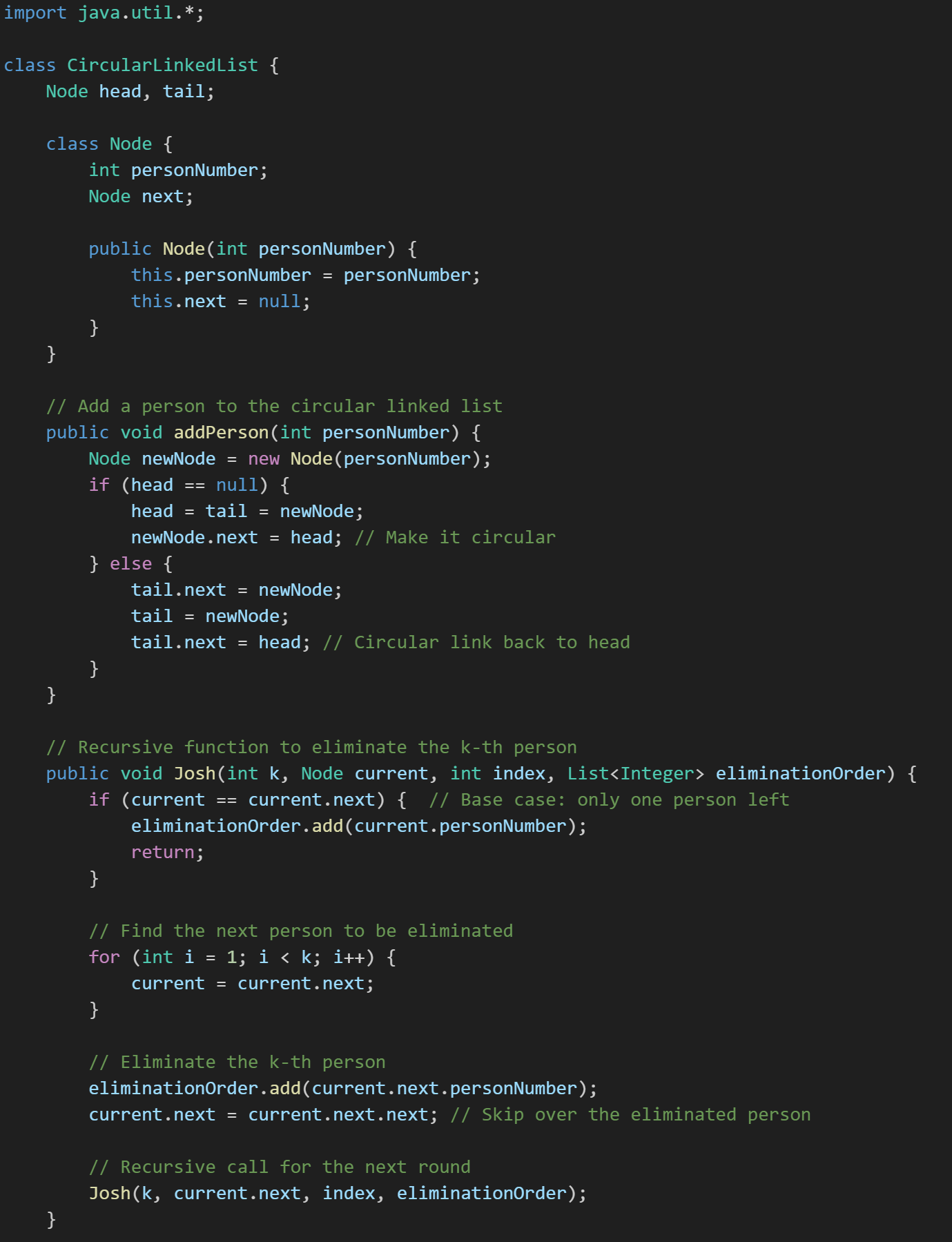
* The Josh() method is a recursive function that eliminates the k-th person. It moves k-1 steps forward and eliminates the person at that position.
* The recursion continues until only one person is left, who is the last person standing.

Start Elimination:

* The startElimination() method first finds the person where counting starts and then invokes the recursive elimination.
* It collects the elimination order and outputs it at the end.

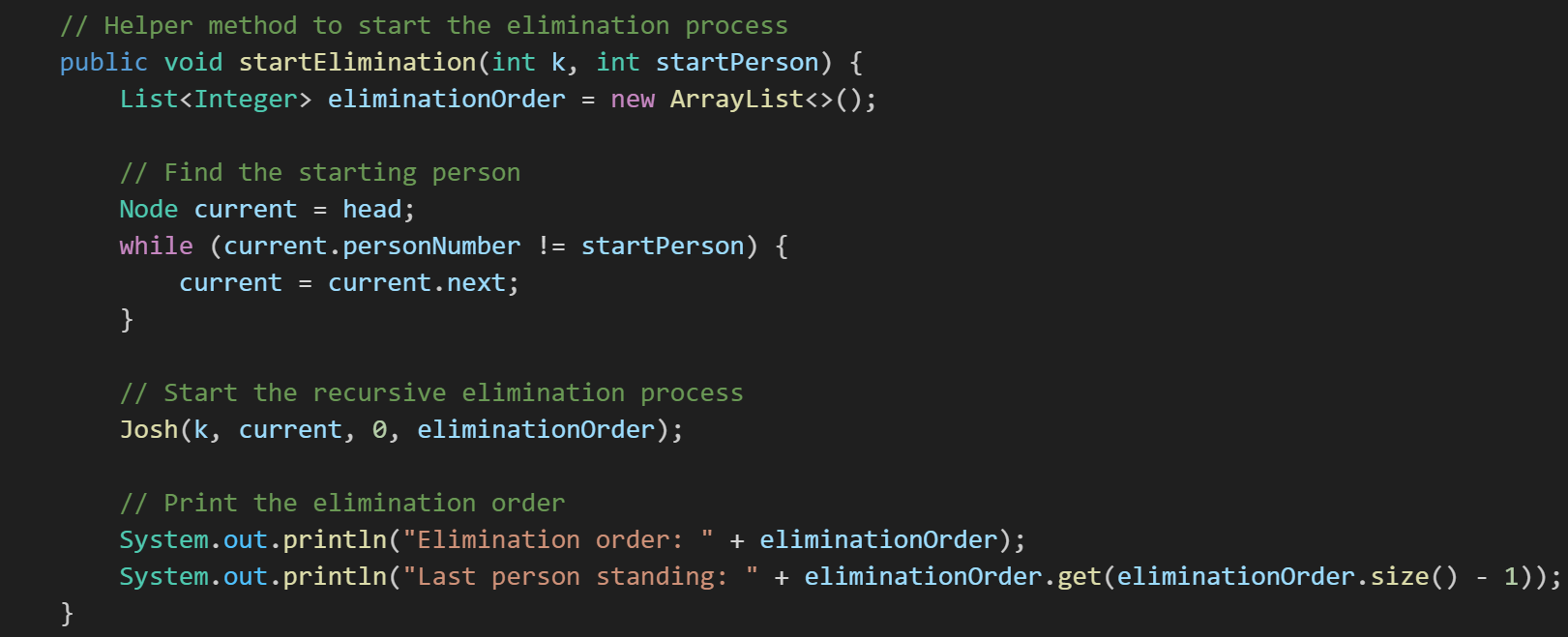
Input:

* The user is prompted to input the total number of people, the counting step (k), and the person where counting begins.



Elimination Order: The list of people eliminated in the order they are killed.

Last Person Standing: The last person remaining after all eliminations



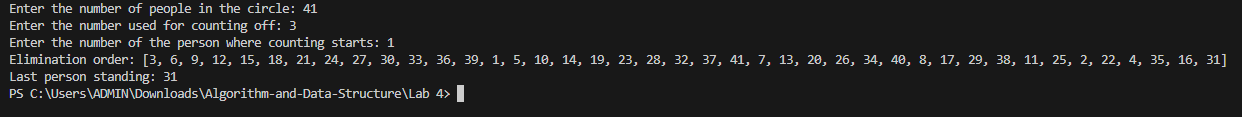
The elimination follows a circular pattern where the counting wraps around when it reaches the end of the list. This is handled by the circular nature of the linked list, where each node points back to the first node after the last one.

Main:

A computer screen shot of code

Description automatically generated

Output:

****