CSCI 241 Data Structures
Project 2 Part II: The Josephus Problem

The story is slightly bloody. During the Jewish-Roman war, Josephus was among a band of 41 Jewish rebels trapped in a cave by the Romans. Preferring suicide to capture, the rebels decided to form a circle and, proceeding around it, to kill every third remaining person until no one was left. But Josephus, along with a friend, wanted none of this suicide nonsense; he quickly calculated where he and his friend should stand in the vicious circle. The end of the story is that both Josephus and his friend lived and Josephus later became a famous historian.

In our variation, we start with n people numbered 1 to n around a circle, and we eliminate every second remaining person until only one survives. The problem is to determine the survivor's number. We first try a small example. Let n = 10. Then the elimination order is 2, 4, 6, 8, 10, 3, 7, 1, 9, so 5 survives.

How to solve the problem? You may pause here and think about the possible algorithm and data structure by yourself. Well, you may have some idea that seems "not so easy", and you may realize you could use either arrays or linked lists as the data structure. Here, we provide you one efficient algorithm to solve the problem using the linked list class that you implemented. The algorithm is smart and simple: **rotate the list to the left by one position circularly, and then delete the first element; repeat it until there is only one element left in the list**. Make sure you understand why.

Once you have your Linked_List class implemented, use it to solve the Josephus problem, for which we have provided a skeleton file. In the main function of this implementation, fill in the `#TODO` section by creating a doubly linked list object called `ll` (2 Ls) with n integers from 1 to n, where n is input by the user of the program. Then call the function `Josephus()` by passing the `ll`. In the function `Josephus()`, fill in the `#TODO` section by using the algorithm described above. Print the sequence of survivors after each death. Finally print the survivor's number.

Below is a run of the code for n = 21 rebels. The sequence of survivors is shown after each death, but rotated a bit. **Your implementation should give the output in the same format for n rebels, where n is any positive integer. Discuss about this algorithm's performance in your writeup.**

Initial order: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 1]
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 1, 3]
[7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 1, 3, 5]
[9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 1, 3, 5, 7]
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 1, 3, 5, 7, 9]
[13, 14, 15, 16, 17, 18, 19, 20, 21, 1, 3, 5, 7, 9, 11]
[15, 16, 17, 18, 19, 20, 21, 1, 3, 5, 7, 9, 11, 13]
[17, 18, 19, 20, 21, 1, 3, 5, 7, 9, 11, 13, 15]
[19, 20, 21, 1, 3, 5, 7, 9, 11, 13, 15, 17]

[21, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
[3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
[7, 9, 11, 13, 15, 17, 19, 21, 3]
[11, 13, 15, 17, 19, 21, 3, 7]
[15, 17, 19, 21, 3, 7, 11]
[19, 21, 3, 7, 11, 15]
[3, 7, 11, 15, 19]
[11, 15, 19, 3]
[19, 3, 11]
[11, 19]
[11]
The survivor is: 11