Logo, company name

Description automatically generated

Assignment 1

Data Structures and Algorithms

Submitted to Dr. Omer Ahmed

Date: 22nd October 2022

# Task:

Your task is to develop a circular doubly linked-list based simulation of the Josephus problem. The simulation will be text based.

The user should be presented with a text-based menu asking him to enter the total number of people (n), starting point (i), direction (clockwise/anti-clockwise) and number to be skipped (k). Your program then must populate a circular linked list with n nodes where data of each node should be their position in the circle (starting from 1), and their name. The names should be read randomly from the list of names provided (names.txt) and no names should be repeated.

Your program should then work iteratively printing the remaining persons after each iteration (round of killing). After the last iteration only the node with the winning initial position should be left in the list.

# Answer:

The initial skeleton code that we must make consists of the following.

* Globally define the Total names as 100 to make our code more convenient.
* 4 files namely, main.c, josephus.c (containing functions), josephus.h (containing prototypes) and node.h (which defines the structures). For node.h we use struct person. This structure saves both the name of the person and also the sequence in which the name is being inserted in the list.
* Utilizing file handling to open and access the files.
* Generating an array which randomly accesses and reads names from the provided file while avoiding repetition of any names.
* Reading the names line by line after taking input for total number of people and inserting them in a doubly circular linked list.
* Asking for direction of rotation, how many people to skip, starting point and then printing people according to the set conditions.

# Problem 1:

Making a non-repeating random array with size 100. Initially the loop was set to make a random array from 1 to 100 but making it non-repeating was difficult.

# Solution:

For this problem, make a loop for every index in the array. Randomly assign values to a temporary integer tmp. Set up a condition that states that if the loop is 0 or if the number is already chosen, then assign another value to tmp. Put the value of tmp in the array for an index i. To tackle the non-repeating part, take some help from stack overflow to make the isAlreadyAdded Function which helps perform the required function. At first, the loop would go from 1 to 100 but then fixed it by making the loop go from 0 to 99 since we are sending index instead of total number of people in the array.

Text

Description automatically generated

A picture containing text

Description automatically generated

Problem 2:

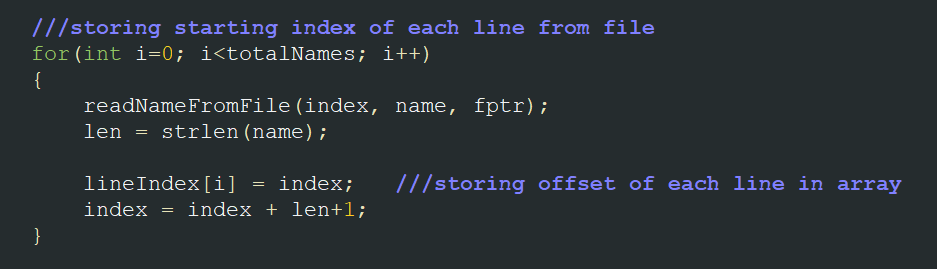
Reading the file line by line. The file contained names of varying length and even names with spaces in between like “Abdul Ahad”. Reading the file line by line was challenging.

Solution:

When reading the file, the assumption made was that when the random array’s first element was sent that the entire name would be read from the file and added to the array but instead only the first character would be added to the first index of the array. Thus, fixed this by seeking to the required index of the file and then using fgets to save into name, with an arbitrary length of 17, chosen to traverse the name i.e., 17 characters will be read from the index specified line. We read the names from the file using the following function. But what we wish to store is actually the indices of said names.

Text

Description automatically generated Using a loop in main to store the indices individually from the file to the array. This is done as follows:

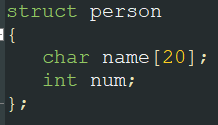


# Problem 3:

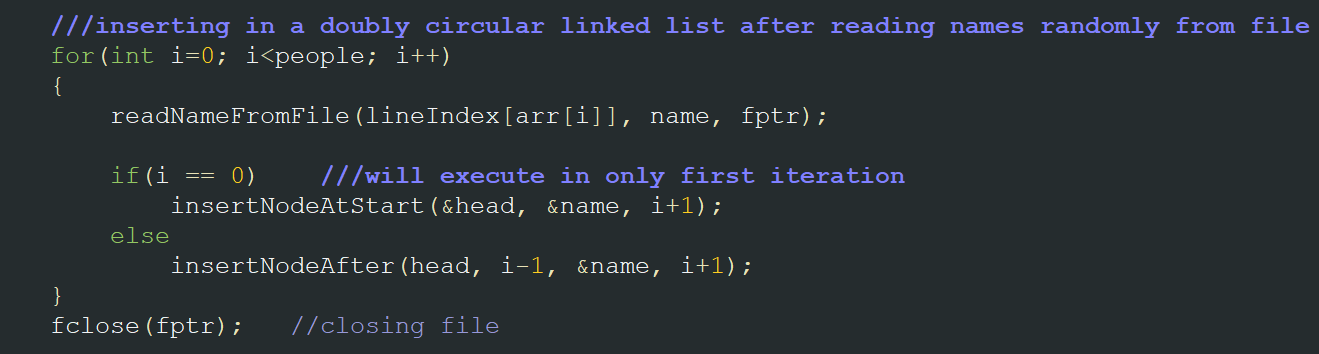
Placing the names from the file into a doubly circular linked list data structure with utilization of the previously made array. Having made an array filled with random indices of names from the file, the next step was to put them in a data structure.

# Solution:

Initially, the names were being saved in the array without sequence. This made finding out which person at which sequence number would win difficult. To solve this condition, add a member to the struct node person called int num. This int num would define the sequence of the name.

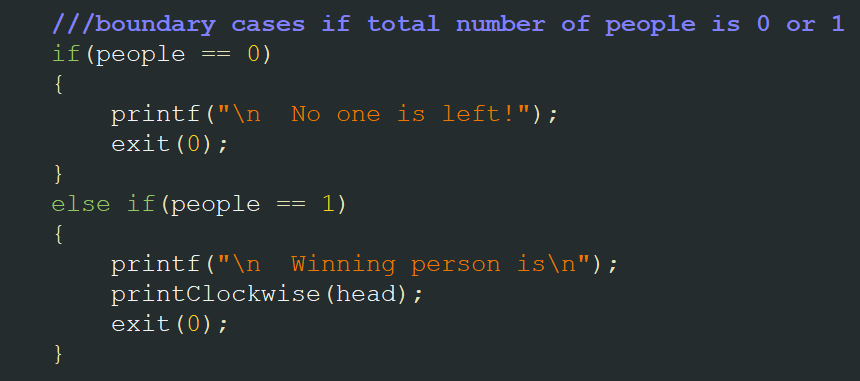


Doubly circular linked list is data structure chosen to tackle this problem. Using the readNameFromFile function to read the lines from the file at the selected index using the random array, simply put them into a doubly circular linked list as follows. Use InsertNodeAtStart function only in the single case when the index of the array is 0, after which use the insertNodeafter function.



# Boundary Cases:

The only boundary cases for this problem are, if the total number of people input by the user is 0 or 1. In which case, either print that no one is left or that the only person that was input is the last remaining person and has thus, won.

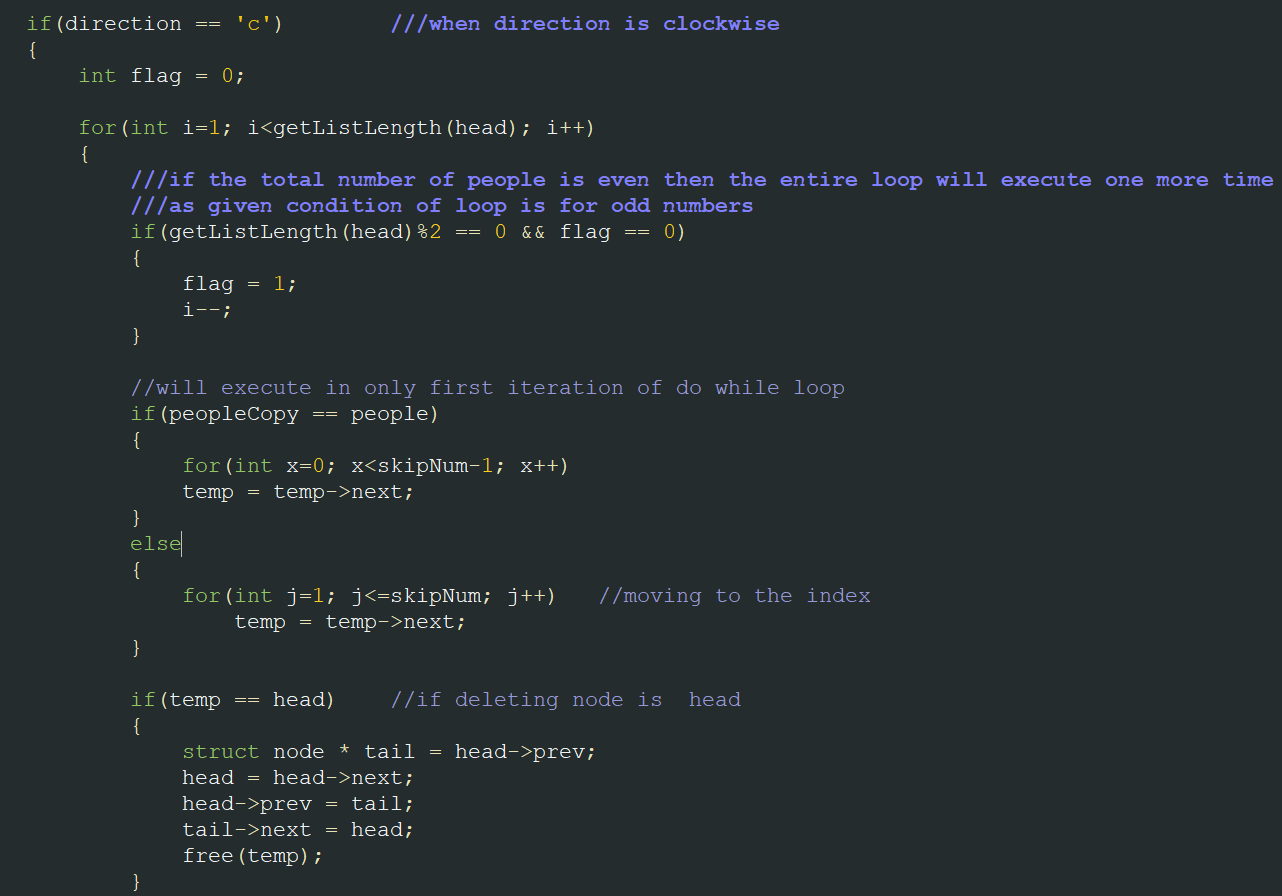


# Problem 4:

Setting Initial Conditions for Josephus problem. This was very mindboggling as initially all the conditions were thought to impede each other. With so many conditions in place, finding where to start working on the problem itself was challenging.

Solution:

We have to tackle the conditions one by one to solve the problem. As in, First we make the basic structure of the Josephus problem while ignoring the conditions. Once we have the basic structure for the problem down, simply integrate the conditions around the basic structure. By tackling the conditions one by one in this manner and dividing the big problem into multiple little problems, tackling those problems becomes much more manageable. To make the problem even simpler divide the conditions into 2 categories clockwise and anti-clockwise respectively. This makes the code more efficient and take less space. For clockwise code we tackle all the conditions like this:



Text

Description automatically generated

For anti-clockwise movement of the circle the tackled conditions are as follows: Text

Description automatically generated

Text

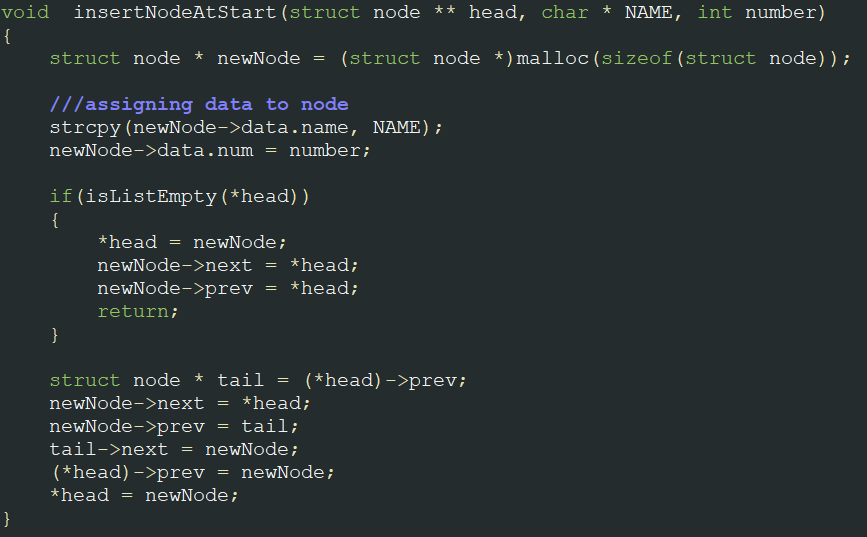
Description automatically generated

Problem 5:

Inadequacy of default insertNodeAtStart and insertNodeAfter. These basic functions are a little tough to use for the problem in their default state. Thus, changing the parameters of these functions is required.

# Solution:

Beyond the basic parameters of the insertNodeAtStart function some other parameters are set up to help ease writing the code. Similarly, rather than tackling all the boundary cases for insertNodeAtStart function, simply use a version of it in which the given index will always be 0 and the list will always be empty since that is the only condition in which this function is being used in the code. The function after modifications is given as under:



Remove some boundary cases for the insertNodeAfter function too as the conditions in which the function to be used are known. As the function is never going to be used in an empty list, remove that condition, and modify the parameters for our ease. The final modified insertNodeAfter is given below:

Text

Description automatically generated

# Output:

