Josephus Problem

Francesca Drăguț

April 2021

1 Design Choices

1.1 Main Ideas

- The elements in the list only have associated keys, which represent their place in the circle, with respect to the head.
- The head is considered to be the element with the key 1.
- For a given node, the node right after it is killed.
- When counting the third element, the current node is counted too.
- Appending is done after the last element before the head.
- Deletion is done by key.
- The implementation contains 2 classes: one for the nodes and one for the list itself.
- In the classes, the variables are private and they are accessed through associated public functions.
- The functions for each class are placed beneath the declaration of the class, so that it is clear they belong to the class above.

1.2 Detailed Description of the Program

For implementing this solution, a circular linked list was used. Each node of the list belongs to the class Node() (lines 5 - 16). Each node has a key, representing the person's place in the circle, with respect to the node that is decided to be the head. Also, each node has a pointer to the next node in the list. Both the key and the pointer to the next are private attributes. To be able to access these variables, the getkey(), getnext(), setkey(), setnext() functions were created, which are public (see lines 12 - 15 and 27 - 41). Also, the constructors of the node are public.

The second implemented class is Circle(). This defines the circular linked list. The private attributes of Circle() are the head of the list and its size. In public, the function that solves the Josephus Problem was defined, along with the functions that Josephus() needed to be solved. Also, in the public area, there are the functions getsize(), increasesize() and decreasesize() (see lines 55 - 57 and 69 - 79). Since the size is private, these functions were needed, so the size can be modified in the functions. The constructors of the class are also public.

The first function, Node * Circle::search(int), searches for a node by a given value (see lines 80 - 97).

The second function, void Circle::appendNode(Node *, int), appends a node to the list (see lines 99 - 114). If the given node is NULL, it means it's the head and it will point to itself. Else, it searches for the last element before the head and places the new node right after it; then, the new node will point to the head. After the addition to the list is done, the size will be increased via the public function increasesize() of the private variable size.
The function void Circle::createList(int) creates the list (see lines 116 - 124). The nodes of the list only have a key, which is the index of the node in the list. Since the nodes do not have a value,

the list is created from 1 to n, where n is the desired size of the circle, entered from the console. For each number in range $(1, n)$, a new node is created, its key is set to i and then it is appended at the end of the list.

The function void Circle::deleteByKey(int) deletes a node by its key (see lines 126 - 160). The first step is to search the node by the desired key. There are several cases here, but they were all treated. The node could be the head and in this case, the list could consist only of the node, or it could have more elements. Or the node could be an arbitrary node from the circle, except for the head. After a deletion operation, the private variable size is decreased by calling the public decreasesize() function.

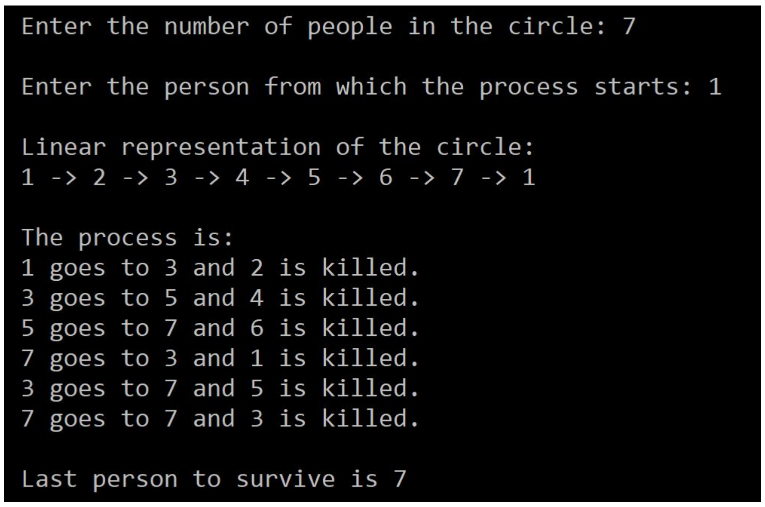
The function void Circle::print() was used to show a linear representation of the circle (see lines 162 - 176). A printing choice was to also show the head again at the end of the list, so that it is more visible that it is a circular linked list.

The function void Circle::thirdNode(Node *) was used to display the third node from a given one (see lines 178 - 187). The implementation presented here kills the next node from the starting node and so on. So when counting the third node, the starting one, or the given one, are considered, meaning that the counting starts from 1. This is a design choice suited for this implementation of the problem.

The last function is the one that actually solves the problem, void Circle::Josephus(int, int) (see lines 189 - 208). The first argument of the function is the starting value and the second one is the size of the list, or the number of people in the circle. There is a node p, which is the node resulted from the search in the list by the key i. A design choice was to also show the process of the killings, which is displayed. While there is more than one node in the list, a node will be deleted by the key of the next element of the current node. But before the deletion occurs, the current node will be linked to the third node (where the first node is considered to be the node itself). Then, the current node will be moved forward by one position and the loop will be retaken. Finally, the surviving node will be displayed.

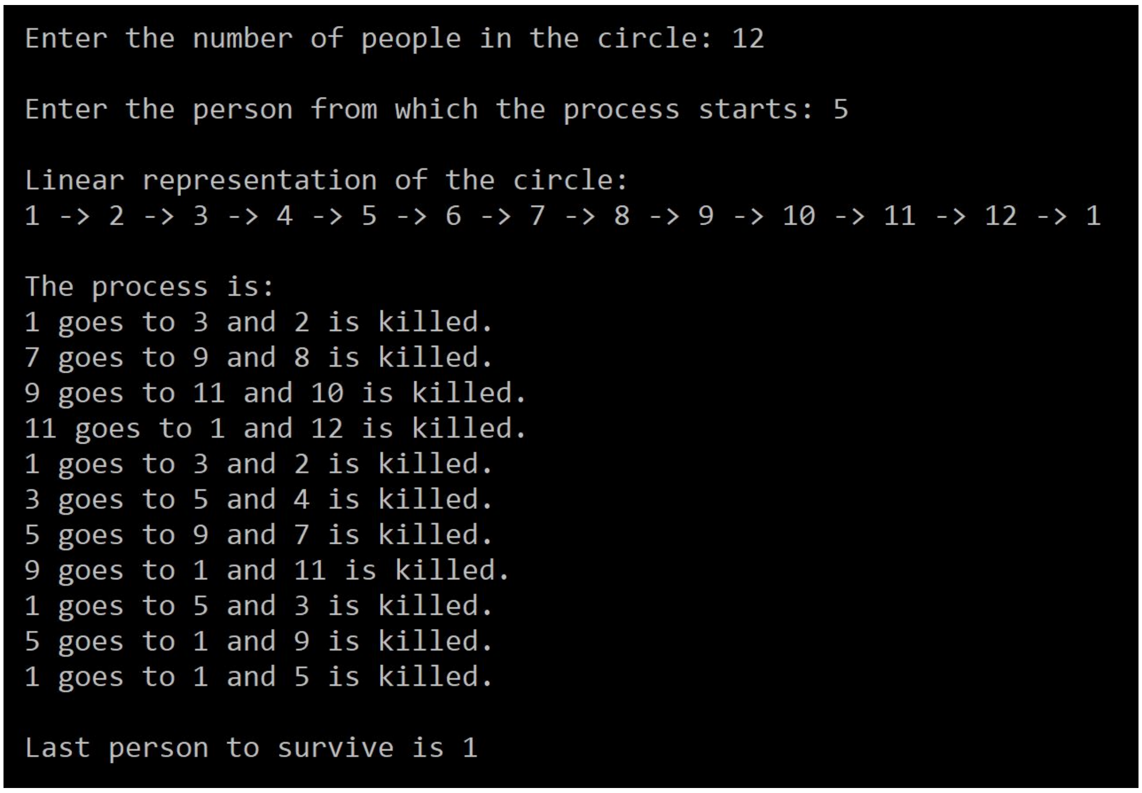
In the main() function (see lines 210 - 227), obj was declared as a member of the class Circle. The console will ask the user for the number of people in the circle and also for the one from which the process should start. Then, the list is created, printed and the problem is solved by calling the function obj.Josephus(i,n);.

2 Experiments



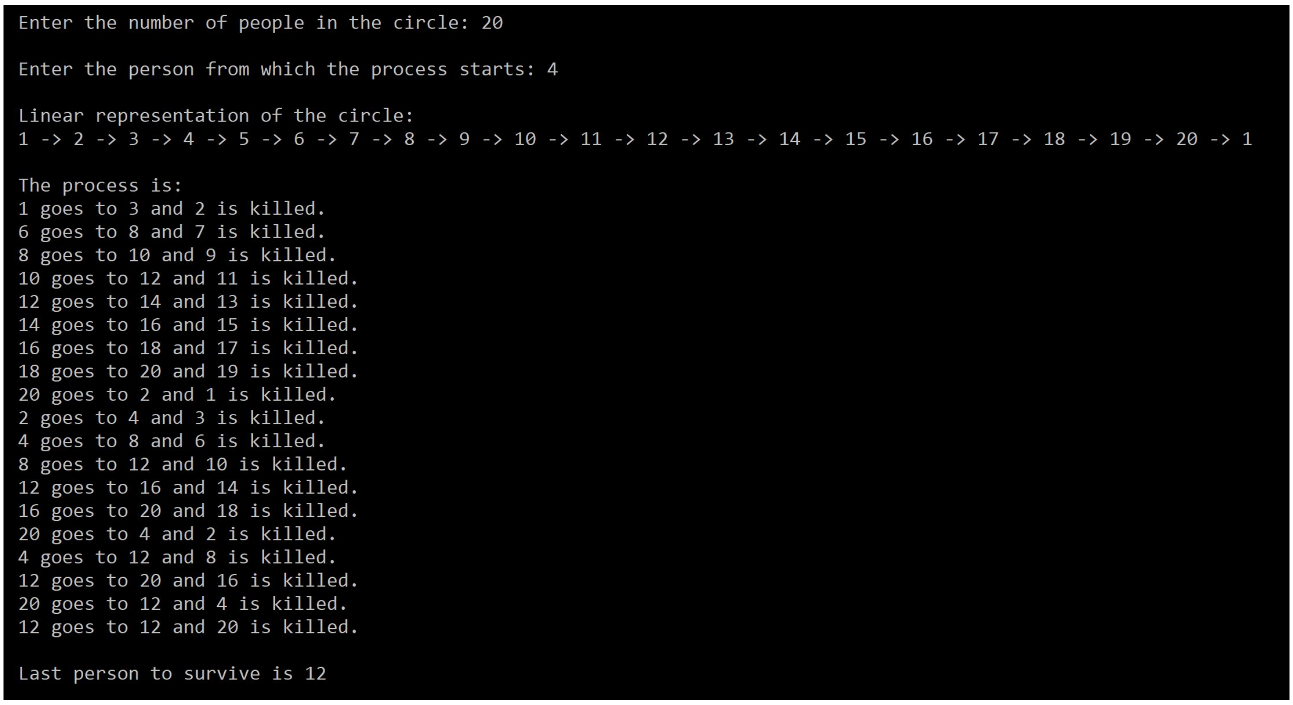
```
Enter the number of people in the circle: 7

Enter the person from which the process starts: 1

Linear representation of the circle:
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 1

The process is:
1 goes to 3 and 2 is killed.
3 goes to 5 and 4 is killed.
5 goes to 7 and 6 is killed.
7 goes to 3 and 1 is killed.
3 goes to 7 and 5 is killed.
7 goes to 7 and 3 is killed.

Last person to survive is 7
```

Figure 1: Example for 7 people, when the process starts from person 1.



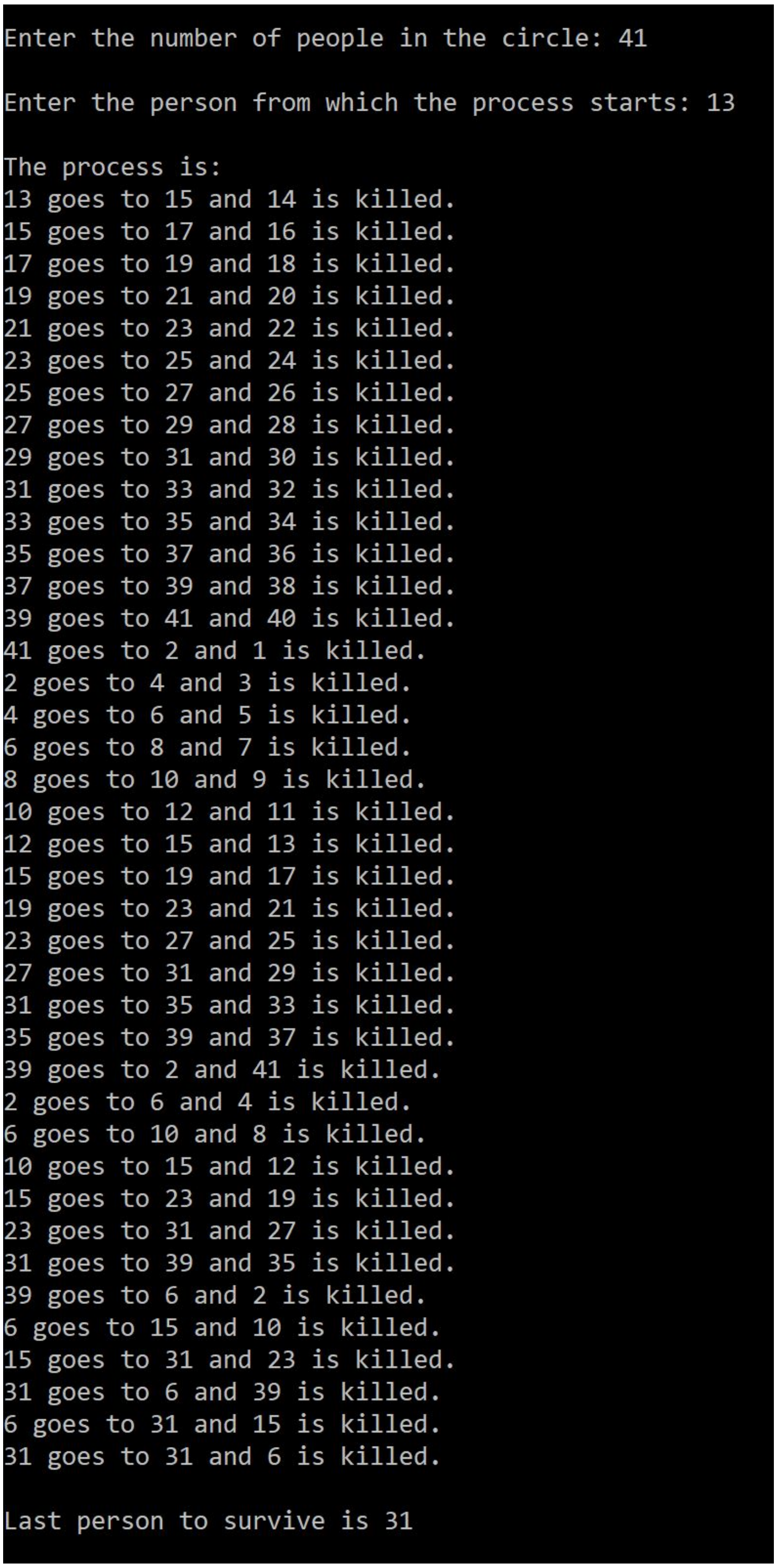
```
Enter the number of people in the circle: 12

Enter the person from which the process starts: 5

Linear representation of the circle:
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 1

The process is:
1 goes to 3 and 2 is killed.
7 goes to 9 and 8 is killed.
9 goes to 11 and 10 is killed.
11 goes to 1 and 12 is killed.
1 goes to 3 and 2 is killed.
3 goes to 5 and 4 is killed.
5 goes to 9 and 7 is killed.
9 goes to 1 and 11 is killed.
1 goes to 5 and 3 is killed.
5 goes to 1 and 9 is killed.
1 goes to 1 and 5 is killed.

Last person to survive is 1
```

Figure 2: Example for 12 people, when the process starts from person 5.



```
Enter the number of people in the circle: 20

Enter the person from which the process starts: 4

Linear representation of the circle:
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 1

The process is:
1 goes to 3 and 2 is killed.
6 goes to 8 and 7 is killed.
8 goes to 10 and 9 is killed.
10 goes to 12 and 11 is killed.
12 goes to 14 and 13 is killed.
14 goes to 16 and 15 is killed.
16 goes to 18 and 17 is killed.
18 goes to 20 and 19 is killed.
20 goes to 2 and 1 is killed.
2 goes to 4 and 3 is killed.
4 goes to 8 and 6 is killed.
8 goes to 12 and 10 is killed.
12 goes to 16 and 14 is killed.
16 goes to 20 and 18 is killed.
20 goes to 4 and 2 is killed.
4 goes to 12 and 8 is killed.
12 goes to 20 and 16 is killed.
20 goes to 12 and 4 is killed.
12 goes to 12 and 20 is killed.

Last person to survive is 12
```

Figure 3: Example for 20 people, when the process starts from person 4.



```
Enter the number of people in the circle: 41

Enter the person from which the process starts: 13

The process is:
13 goes to 15 and 14 is killed.
15 goes to 17 and 16 is killed.
17 goes to 19 and 18 is killed.
19 goes to 21 and 20 is killed.
21 goes to 23 and 22 is killed.
23 goes to 25 and 24 is killed.
25 goes to 27 and 26 is killed.
27 goes to 29 and 28 is killed.
29 goes to 31 and 30 is killed.
31 goes to 33 and 32 is killed.
33 goes to 35 and 34 is killed.
35 goes to 37 and 36 is killed.
37 goes to 39 and 38 is killed.
39 goes to 41 and 40 is killed.
41 goes to 2 and 1 is killed.
2 goes to 4 and 3 is killed.
4 goes to 6 and 5 is killed.
6 goes to 8 and 7 is killed.
8 goes to 10 and 9 is killed.
10 goes to 12 and 11 is killed.
12 goes to 15 and 13 is killed.
15 goes to 19 and 17 is killed.
19 goes to 23 and 21 is killed.
23 goes to 27 and 25 is killed.
27 goes to 31 and 29 is killed.
31 goes to 35 and 33 is killed.
35 goes to 39 and 37 is killed.
39 goes to 2 and 41 is killed.
2 goes to 6 and 4 is killed.
6 goes to 10 and 8 is killed.
10 goes to 15 and 12 is killed.
15 goes to 23 and 19 is killed.
23 goes to 31 and 27 is killed.
31 goes to 39 and 35 is killed.
39 goes to 6 and 2 is killed.
6 goes to 15 and 10 is killed.
15 goes to 31 and 23 is killed.
31 goes to 6 and 39 is killed.
6 goes to 31 and 15 is killed.
31 goes to 31 and 6 is killed.

Last person to survive is 31
```

Figure 4: Example for 41 people, when the process starts from person 13. This might have been the case of Josephus, who is said to have been placed at the index 31.

3 Bibliograhpy

The main sources were the slides from class and the book by Cormen.
As other sources and further reading, the following resources were used. [1] and [2] were used to find out more about the Josephus Problem and its different versions and shapes. They were also used to understand the mathematical part of the problem. [3], [4] and [5] were used for the programming part.

References

[1] Thomas Cormen. The Feline Josephus Problem. *Theory Comput. Syst.*, (50):20–34, 2012.

[2] Peter Henderson. The Josephus Flavius' problem. *SIGCSE Bulletin*, (38):17–20, 2006.

[3] D. Woodhouse. Programming the Josephus problem. *ACM Sigcse Bulletin*, (10):56–58, 1978.

[4] G. Gonnet. *Handbook of Algorithms and Data Structures.* Addison - Wesley Publishing Company, 1991.

[5] Adam Drozdek. *Data structures and algorithms in C++. Fourth edition.* Cenage Learning, 2004.