ADS - The Josephus Problem using a Circular list

Cătălin Chiriță

May 2020

1 Introduction

First i started by researching the Josephus problem.The problem is named after Flavius Josephus, a Jewish historian living in the 1st century. According to Josephus' account of the siege of Yodfat, he and his 40 soldiers were trapped in a cave by Roman soldiers. They chose suicide over capture, and settled on a serial method of committing suicide by drawing lots. Josephus states that by luck or possibly by the hand of God, he and another man remained until the end and surrendered to the Romans rather than killing themselves. This is the story given in Book 3, Chapter 8, part 7 of Josephus' The Jewish War (writing of himself in the third person). Which to be honest is a pretty stupid solution to a problem but stupid problems require stupid solutions.

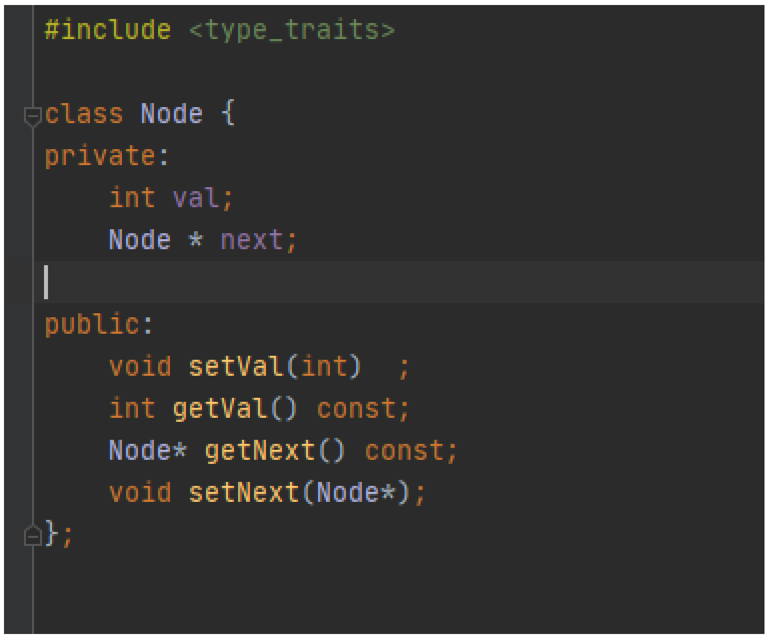
2 The Algorithm

"At first I was afraid, I was petrified"

- Me after reading the project description

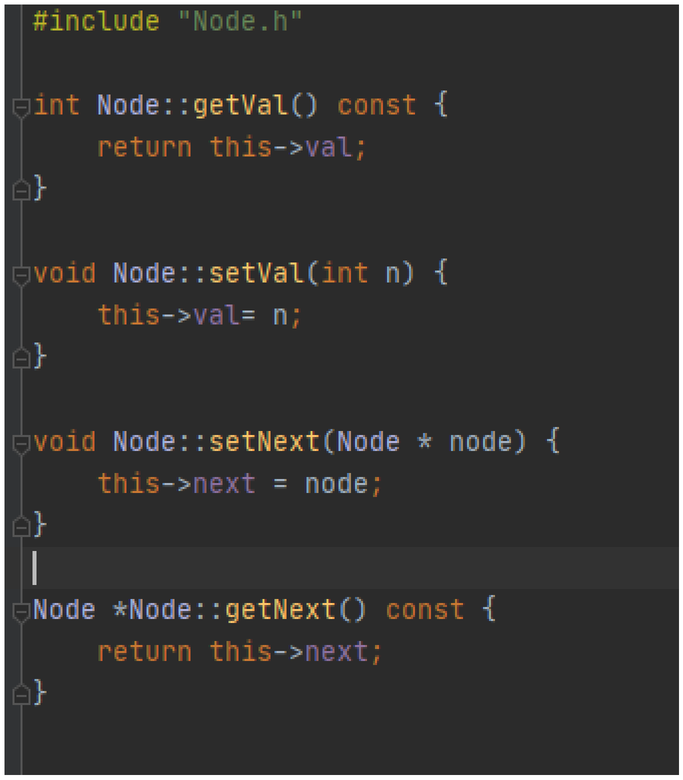
2.1 The Node

It happened suddenly, as i booted CLion inspiration hit me. I made the solution to the problem by in C++. I created two classes: one for the nodes and for the list itself, which is linked simply and circular. The nodes represent the poor soldiers from the problem, having two private variables: a value (int) and a node called next that will connect it to another node.



```cpp
#include <type_traits>

class Node {
private:
    int val;
    Node * next;

public:
    void setVal(int)  ;
    int getVal() const;
    Node* getNext() const;
    void setNext(Node*);
};
```

For the function itself i made 2 get and 2 set for the values and reference.

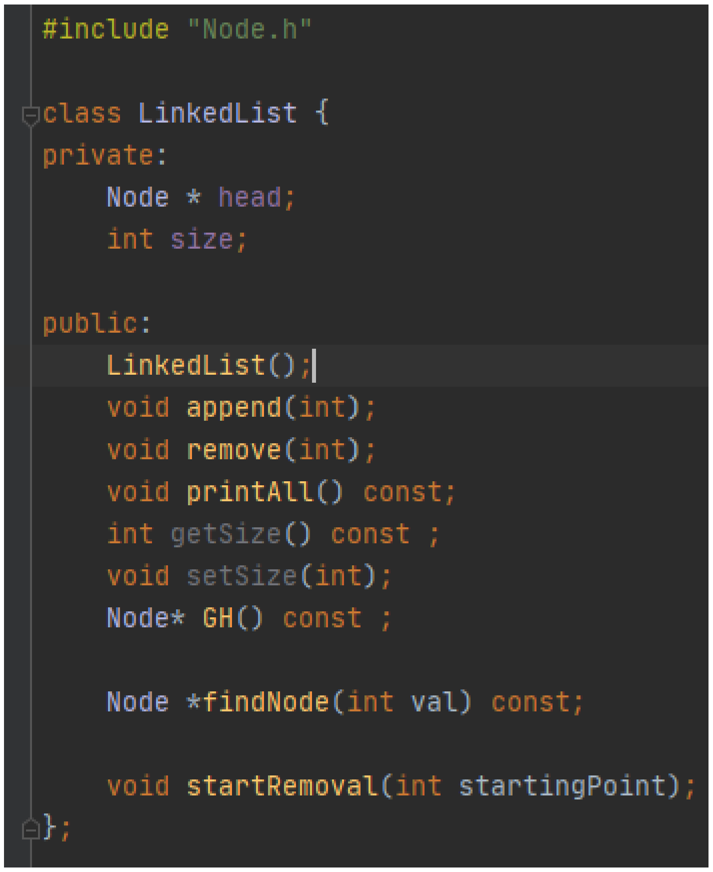


```cpp
#include "Node.h"

int Node::getVal() const {
    return this->val;
}

void Node::setVal(int n) {
    this->val= n;
}

void Node::setNext(Node * node) {
    this->next = node;
}

Node *Node::getNext() const {
    return this->next;
}
```

2.2 The List

- **The Header**

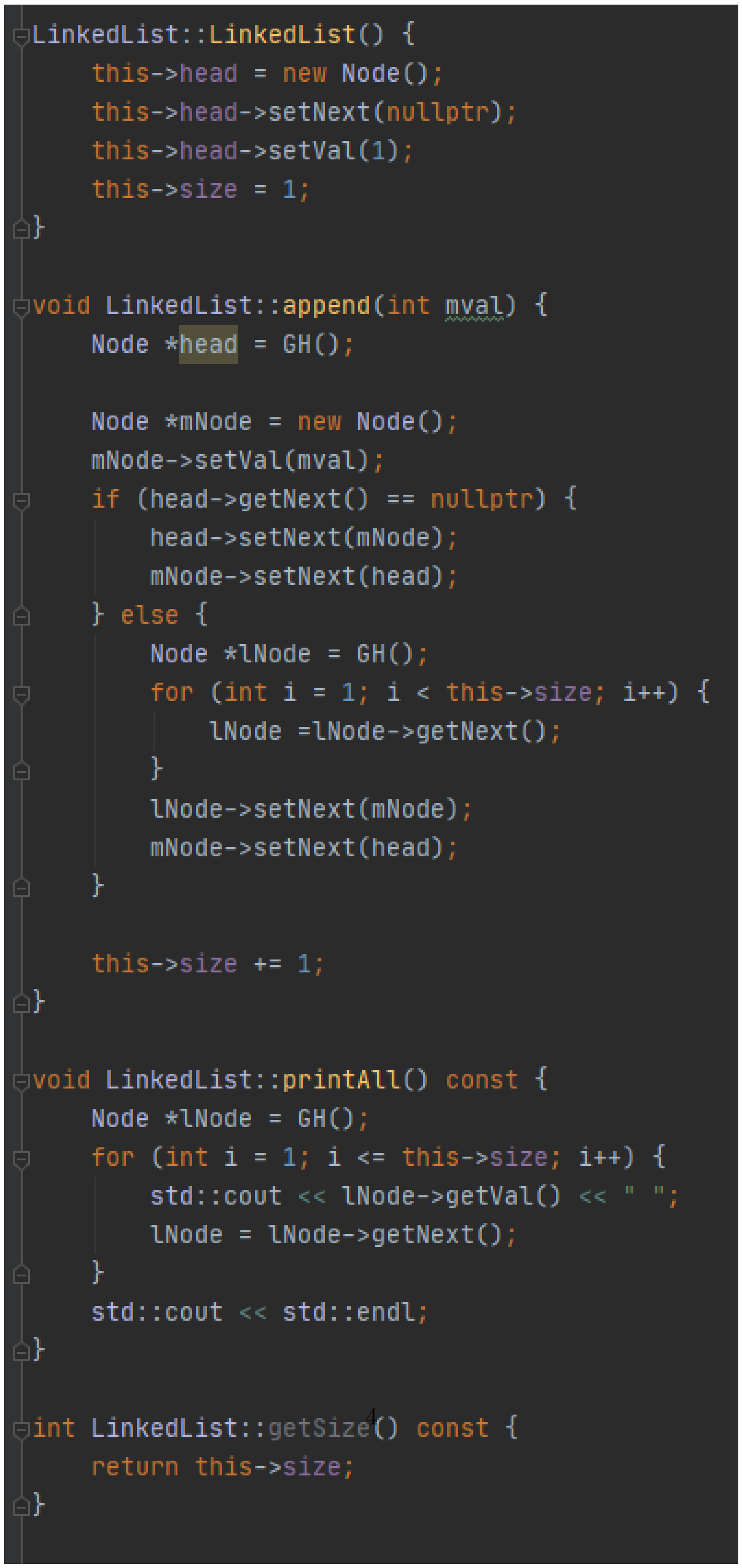
After that, i made the linked list itself, with two variables: the first node called head, of type Node*, and an int for the list's size. Next was to implement the required functions like adding a new node(append), separate one for the removal, printing the list for testing, getting the head of the list and finding a specific node given its value.



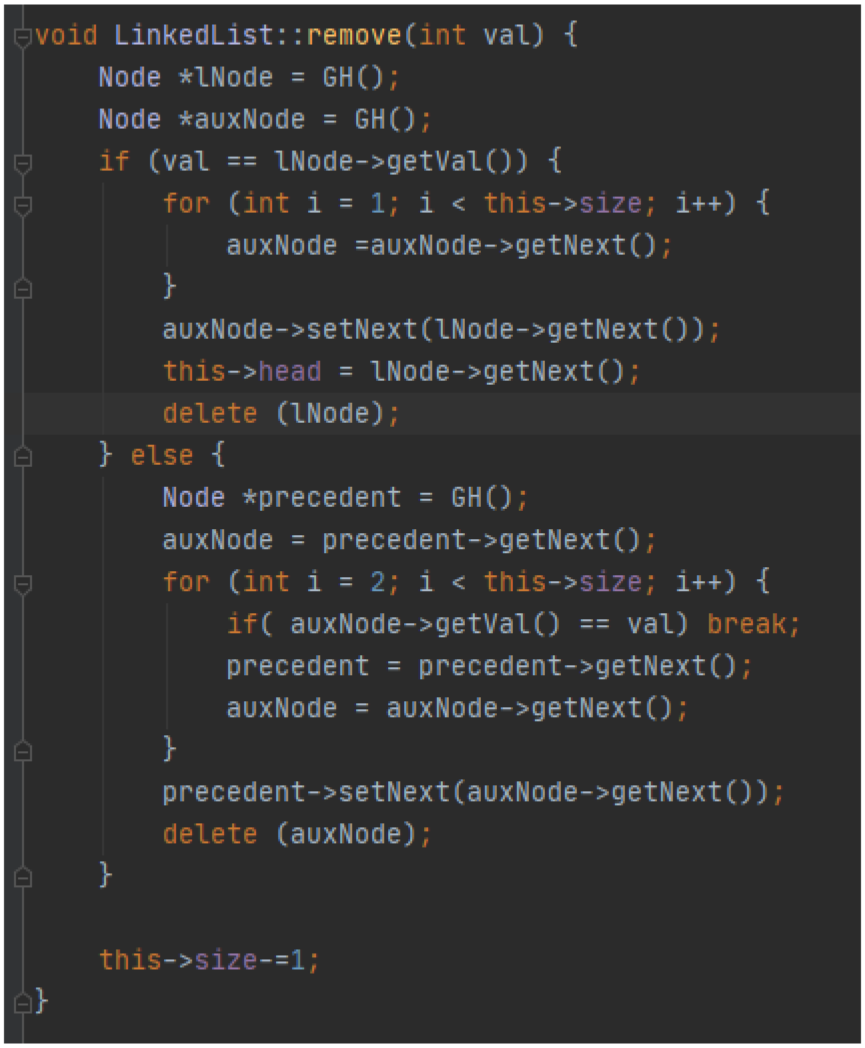
```
#include "Node.h"

class LinkedList {
private:
    Node * head;
    int size;

public:
    LinkedList();
    void append(int);
    void remove(int);
    void printAll() const;
    int getSize() const ;
    void setSize(int);
    Node* GH() const ;

    Node *findNode(int val) const;

    void startRemoval(int startingPoint);
};
```

- **The Functions**

The constructor starts the new list with a head, that i gave the value as 1 from the start. The append function adds a new node to the end of the list using a for to get tthere and will create a new link. It also checks the special case if there is only one node in the list, then the for is no longer needed. Printing was easy to implement. Starts from the head, uses *for*, it prints every value

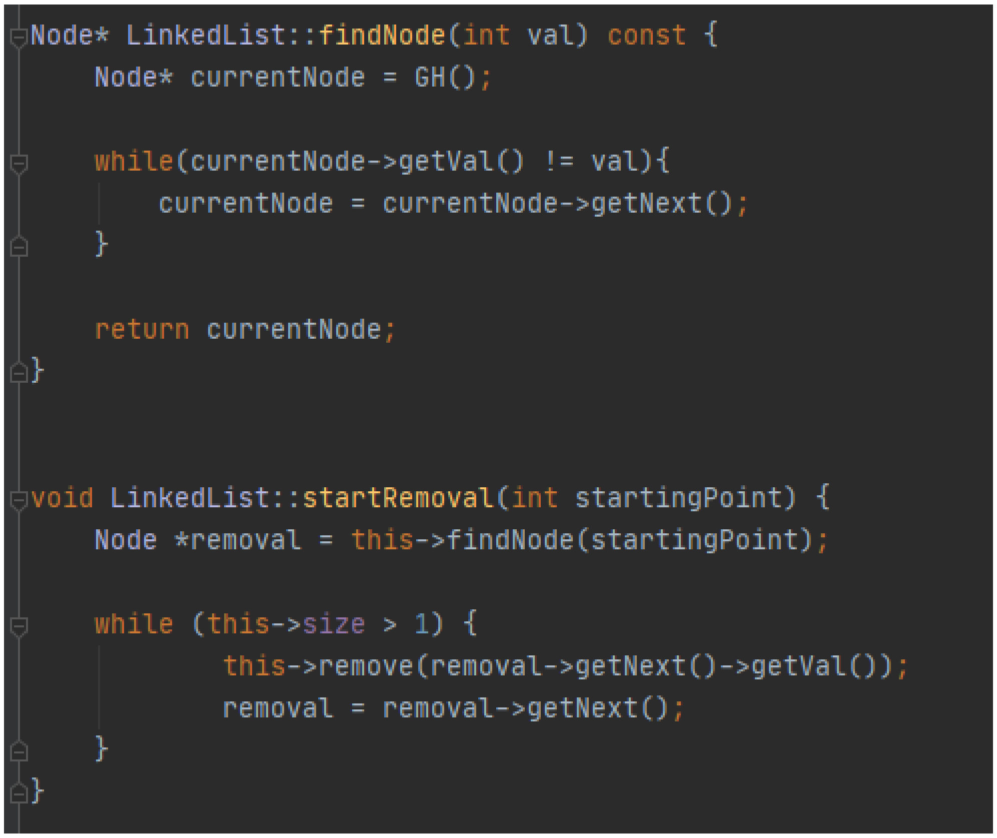

```cpp
LinkedList::LinkedList() {
    this->head = new Node();
    this->head->setNext(nullptr);
    this->head->setVal(1);
    this->size = 1;
}

void LinkedList::append(int mval) {
    Node *head = GH();

    Node *mNode = new Node();
    mNode->setVal(mval);
    if (head->getNext() == nullptr) {
        head->setNext(mNode);
        mNode->setNext(head);
    } else {
        Node *lNode = GH();
        for (int i = 1; i < this->size; i++) {
            lNode =lNode->getNext();
        }
        lNode->setNext(mNode);
        mNode->setNext(head);
    }

    this->size += 1;
}

void LinkedList::printAll() const {
    Node *lNode = GH();
    for (int i = 1; i <= this->size; i++) {
        std::cout << lNode->getVal() << " ";
        lNode = lNode->getNext();
    }
    std::cout << std::endl;
}

int LinkedList::getSize() const {
    return this->size;
}
```

Deleting a node from the tricky part. The function takes an int val as parameter, finds the node with that value, frees the memory and redoes the connections between the nodes. The function checks if the current node has the required value and if not, it finds the right one.



```cpp
void LinkedList::remove(int val) {
    Node *lNode = GH();
    Node *auxNode = GH();
    if (val == lNode->getVal()) {
        for (int i = 1; i < this->size; i++) {
            auxNode =auxNode->getNext();
        }
        auxNode->setNext(lNode->getNext());
        this->head = lNode->getNext();
        delete (lNode);
    } else {
        Node *precedent = GH();
        auxNode = precedent->getNext();
        for (int i = 2; i < this->size; i++) {
            if( auxNode->getVal() == val) break;
            precedent = precedent->getNext();
            auxNode = auxNode->getNext();
        }
        precedent->setNext(auxNode->getNext());
        delete (auxNode);
    }

    this->size-=1;
}
```

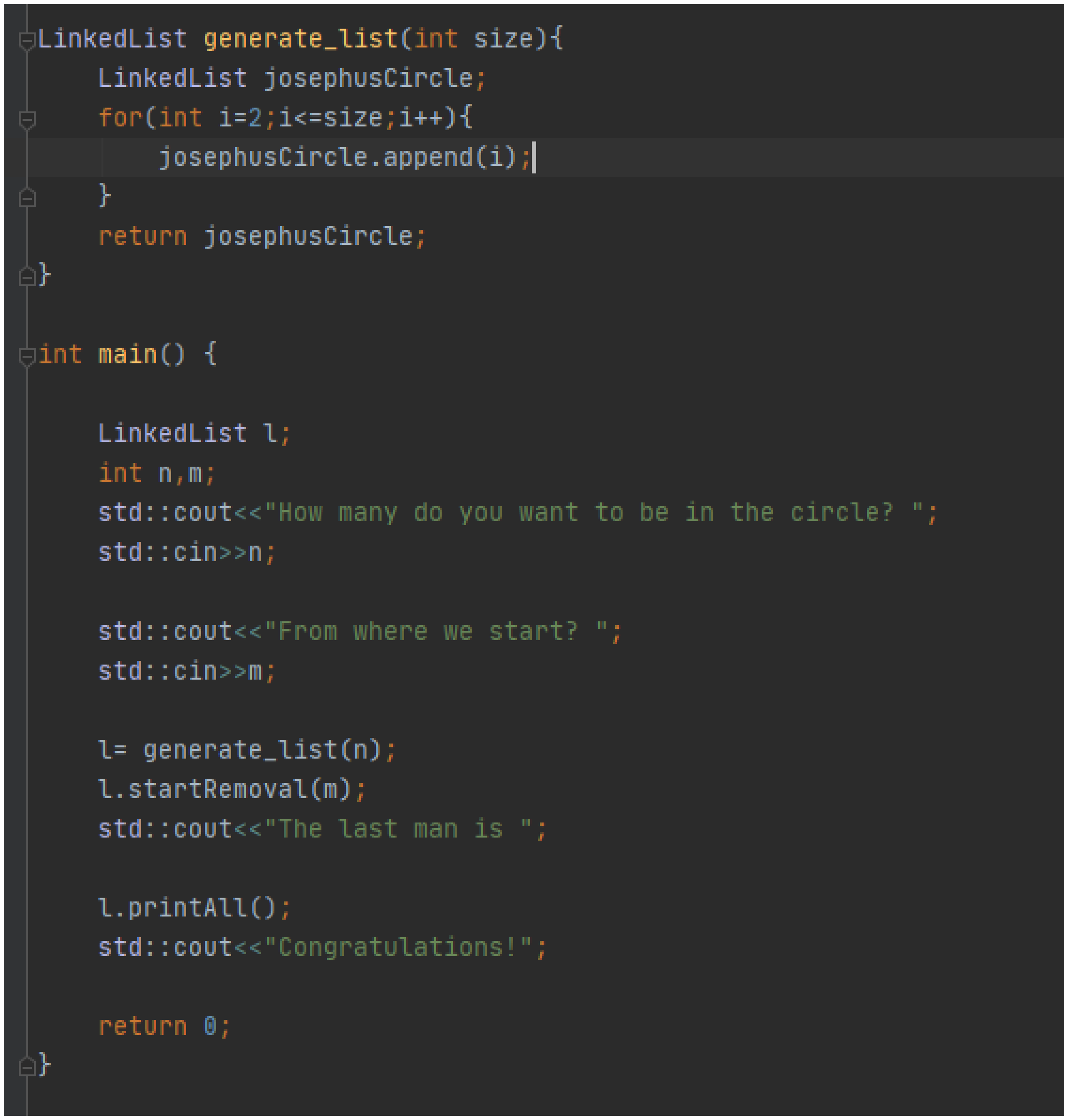
Now for the purpose of the problem: finding out the last remaining. To do this i created a function with an int parameter from where the removal start. To aid us we have a function called findNode that returns the node where we want to start from. Then, I remove the nodes, until there is only one left.



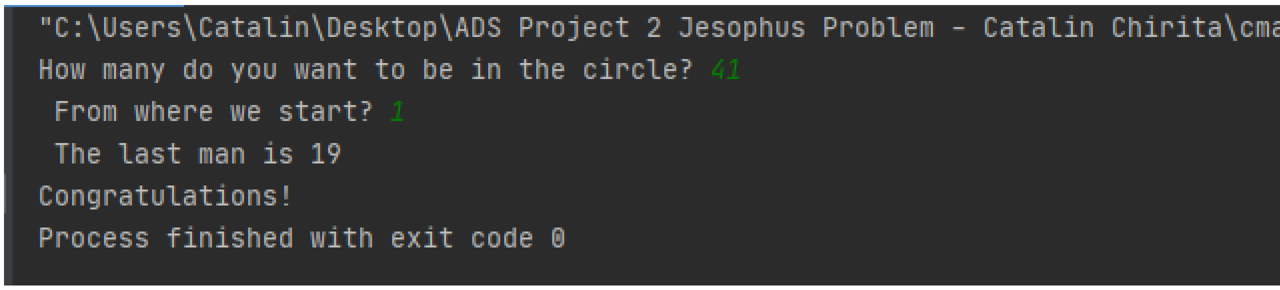
```
Node* LinkedList::findNode(int val) const {
    Node* currentNode = GH();

    while(currentNode->getVal() != val){
        currentNode = currentNode->getNext();
    }

    return currentNode;
}


void LinkedList::startRemoval(int startingPoint) {
    Node *removal = this->findNode(startingPoint);

    while (this->size > 1) {
            this->remove(removal->getNext()->getVal());
            removal = removal->getNext();
    }
}
```

2.3 Main

Here i made a function that generates a list with a n number of nodes, the n is read from the keyboard. Then we introduce the parameters and generate a solution



```cpp
LinkedList generate_list(int size){
    LinkedList josephusCircle;
    for(int i=2;i<=size;i++){
        josephusCircle.append(i);
    }
    return josephusCircle;
}

int main() {

    LinkedList l;
    int n,m;
    std::cout<<"How many do you want to be in the circle? ";
    std::cin>>n;

    std::cout<<"From where we start? ";
    std::cin>>m;

    l= generate_list(n);
    l.startRemoval(m);
    std::cout<<"The last man is ";

    l.printAll();
    std::cout<<"Congratulations!";

    return 0;
}
```



```
"C:\Users\Catalin\Desktop\ADS Project 2 Jesophus Problem - Catalin Chirita\cma
How many do you want to be in the circle? 41
 From where we start? 1
 The last man is 19
Congratulations!
Process finished with exit code 0
```