ADS Project
-Josephus Problem-

Context:

There are n people standing in a circle waiting to be executed. The counting out begins at some point in the circle and proceeds around the circle in a fixed direction. In each step, a certain number of people are skipped and the next person is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom. Which is the last one to survive?

Implementation:

In **" ClinkedList.h"** I created 2 classes Node and ClinkedList with private data such that we need to use the methods get and set in order to access and work with the attributes.

Function **void create_Josephus_circle()** uses a circular linked list with n nodes(each with a key), the list is created from a given node i that becomes the head.

If we start from the first node(key=1) , our Circular Linked List (CLL) looks like this for a given n=7:

```
[1]->[2]->[3]->[4]->[5]->[6]->[7]->
```

If we start from the 5th node(key=5) , our Circular Linked List (CLL) looks like this for a given n=7 :

```
[5]->[6]->[7]->[1]->[2]->[3]->[4]->
```

After the circle is created from the chosen starting node , we need to decide how many people are skipped each loop(variable k is used).

Function **int Josephus_for_k(int k)** returns the position of the person that survives.In this function, we use two special objects from the class Node(**del** and **current**) in order to track the position that we are in, the nodes that corresponds to del are deleted with the function **void delete_node(int k).**

```
How many people are skipped?0

[1]->[2]->[3]->[4]->

We kill 2th person :
[1]->[3]->[4]->

We kill 4th person :
[1]->[3]->

We kill 3th person :
[1]->

Stays alive:1
```

If we do not skip people (k=0) and we start from one (head=1) then the first person stays alive after the people on the positions 2,4,3 are killed in this order.

If we choose k>0 the the **for loop** will move current and del so that we find the (k+1)th node to delete.

The function ends when we are left with only one node, such that:
head->getNext()==head

```
for(int i = 1; i < k; i++)
{
    current = current->getNext();
    del = del->getNext();
}
```

Complexity:

The complexity of the program is O(n), where n is the number of people in the circle.

Observations:

Observation 1:

We can see a pattern in the Josephus problem, but only for k=0(no skipped people).
If we indexed the positions of the people and we start from 1 , all the time people in the even positions will die. The indexing starts over when the circle is finished from the current node.

Step 1:Indexed the people(a->i)

a[1] -> b[2]-> c[3]->d[4]->e[5]->f[6]->g[7]->h[8]->i[9]->

Step 2:Kill the even numbers

a[1]->c[3]->e[5]->g[7]->i[9]->

Step 3: Index again from the current one, **i**'s turn

i[1]->a[2]->c[3]->e[4]->g[5]->

Step 4: Kill again

a[1]->c[3]->g[5]->

Step 5: Index again from the current one, **g**'s turn

g[1]->a[2]->c[3]->

Step 6:Kill again

g[1]->c[3]->

Step 7:Index again form the current one,**c**'s turn

c[1]->g[2]->

Step 8:Kill the last one even g[2].

Conclusion: "**c**" stays alive and all the even indexing people died

Observation 2:

For k=0(no skipped people), every time n is a power of two(2,4,8,16,..), the one who starts is the one who stays alive.

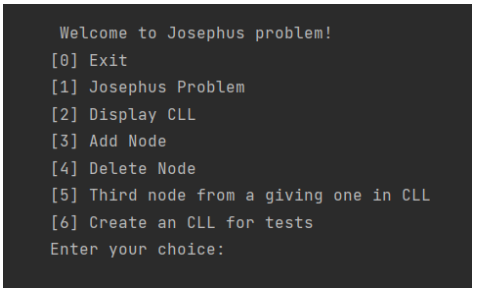
Just a personal dilemma:

I found different interpretations of the Josephus problem, most of them are without any skipped people(k=0), initially I implemented it like that, and for me, is more logical because when we have less people that the people that are going to be skipped we just need to start counting the same nodes multiple times for one kill.

For example, let's take n=4 and k=3, we have the list 1,2,3,4, we start from 1, and we kill 4, then we have 1,2,3 and the current position is back to 1, but now we need to go 1-2-3-1 and kill 1,somehow, the 1 is killing himself if the person in the current position needs to be the killer and we do not want that.

Remark about my code:

In the **"main.cpp"** I used a switch() in order to test my functions, BUT if you want to solve the Josephus problem use only the Option [1] : Josephus problem, because if you create a list [6] and then go to [1] is possible



that the two CLL that are created to interfere and create an error.
If you want to check the Options [2]->[6], please run the code again without choosing the Option [1].