CSE 4304-Data Structures Lab. Winter 2022
Date: August 30, 2022
Target Group: All
Topic: Queues

Instructions:

- Task naming format: fullID_T01L04_1A.c/CPP
- If you find any issues in problem description/test cases, comment in the google classroom.
- If you find any test case that is tricky that I didn't include but others might forget to handle, please comment! I'll be happy to add.
- Use appropriate comments in your code. This will help you to easily recall the solution in the future.
- Obtained marks will vary based on the efficiency of the solution.
- Do not use <bits/stdc++.h> library, add the necessary libraries one after another.

Group	**Tasks**
1B	1 2 3 8
1A	2 5 6 8

<u>Task-01</u>: Implementing the basic operations of Queue.

Implement the following operations with a circular queue.

- void Enqueue (int x)
- int Dequeue()
- int size
- void front
- Void rear
- Bool isEmpty()
- Bool isFull()

Test the functions by your self made test-cases.

Task 02:

You have a class which counts the number of recent requests within a certain time frame.

Implement the function:

- int ping(int t) Adds a new request at time t, where t represents some time in milliseconds, and returns the number of requests that has happened in the past 3000 milliseconds (including the new request). Specifically, return the number of requests that have happened in the inclusive range [t - 3000, t].

It is guaranteed that every call to ping uses a strictly larger value of t than the previous call.

1. *Copy the code into your IDE.*
2. *Edit the **ping(int t)** function.*
3. *Don't touch the **main()** function.*

```cpp
#include <iostream>

using namespace std;

int ping(int t) {
    // YOUR CODE HERE

}

int main() {
    cout << ping(1) << "\n";
    cout << ping(2) << "\n";
    cout << ping(3) << "\n";
    cout << ping(4) << "\n";
    cout << ping(3001) << "\n";
    cout << ping(3002) << "\n";
    cout << ping(3003) << "\n";
    cout << ping(6003) << "\n";
    cout << ping(10003) << "\n";

    return 0;
}
```

Expected Output:

```
1
2
3
4
5
5
5
2
1

Process returned 0 (0x0)   execution time : 0.021 s
Press any key to continue.
```

Task 03:

Description: Implement a **last-in-first-out (LIFO) stack** using queues. The implemented stack should support all the functions of a normal stack (**push, top, pop,** and **empty**).

Implement the Functions:

- void push(int x) Pushes element x to the top of the stack.
- int pop() Removes the element on the top of the stack and returns it.
- int top() Returns the element on the top of the stack.
- boolean empty() Returns true if the stack is empty, false otherwise.

Notes:

- You must use only standard operations of a queue, which means that only **push to back**, **peek/pop from front**, **size** and **is_empty** operations are valid.

Test case: [Note: Running the basic stack operations successfully on *main()* will suffice]

```
#include <iostream>

using namespace std;

// Push into the Stack
void push_s(int x) {

}

// Removes the element on top of the stack.
void pop_s() {
}

// Get the top element.
int top_s() {

}

// Return whether the stack is empty.
bool empty_s() {

}

int main() {
    push_s(10);
    cout << top_s() << endl;
    push_s(20);
    cout << top_s() << endl;
    pop_s();
    cout << top_s() << endl;
    push_s(100);
    cout << top_s() << endl;
    cout << empty_s() << endl;
    pop_s();
    pop_s();
    cout << empty_s() << endl;
}
```

Expected Output:

```
10
20
10
100
0
1

Process returned 0 (0x0)   execution time : 0.051 s
Press any key to continue.
```

Task 05: Time Needed to Buy Tickets

There are n people in a line queuing to buy tickets, where the 0-th person is at the front of the line and the (n-1)-th person is at the back of the line. You are given a 0-indexed integer array tickets of length n where the number of tickets that the i-th person would like to buy is tickets[i].

Each person takes exactly 1 second to buy a ticket. A person can only buy 1 ticket at a time and has to go back to the end of the line (which happens instantaneously) in order to buy more tickets. If a person does not have any tickets left to buy, the person will leave the line.

First line of input contains the info of the n-person in the line (ends with -1). Next line contains the value of k. Return the **time taken for the person at position k** (0-indexed) to finish buying tickets.

Input	**Output**
2 3 2 -1 2	6 Explanation: - First pass, everyone in the line buys a ticket and the line becomes [1, 2, 1]. - Second pass, everyone in the line buys a ticket and the line becomes [0, 1, 0]. The person at position 2 has successfully bought 2 tickets and it took 3 + 3 = 6 seconds.
2 4 3 -1 2	8
5 1 1 1 -1 0	8 Explanation: - In the first pass, everyone in the line buys a ticket and the line becomes [4, 0, 0, 0]. - In the next 4 passes, only the person in position 0 is buying tickets. The person at position 0 has successfully bought 5 tickets and it took 4 + 1 + 1 + 1 + 1 = 8 seconds.
1 5 7 3 1 3 2 -1 3	15 States: 1 5 7 3 1 3 2 0 4 6 2 0 2 1 = 7 units 0 3 5 1 0 1 0 = 5 units 0 2 4 0 0 0 0 = 3 units

Note: Use a queue to solve this problem

Task 06: Number of Students Unable to Eat Lunch

The school cafeteria offers circular and square shaped sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwiche.

The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a stack. At each step:

- If the student at the front of the queue prefers the sandwich on the top of the stack, they will take it and leave the queue.
- Otherwise, s/he will leave it and go to the end of the queue.

This continues until none of the queue students want to take the top sandwich and are thus **unable to eat**.
First line of input will contain the number of students. Then you are given two integer arrays students and sandwiches where sandwiches[i] is the type of the i-th sandwich in the stack (i = 0 is the top of the stack) and students[j] is the preference of the j-th student in the initial queue (j=0 is the front of the queue). Return the number of students that are unable to eat.

Test case:

Input	Output

4
1 1 0 0
0 1 0 1
	0
	Explanation:
	- Front student leaves the top sandwich and returns to the end of the line making students = [1,0,0,1].
	- Front student leaves the top sandwich and returns to the end of the line making students = [0,0,1,1].
	- Front student takes the top sandwich and leaves the line making students = [0,1,1] and sandwiches = [1,0,1].
	- Front student leaves the top sandwich and returns to the end of the line making students = [1,1,0].
	- Front student takes the top sandwich and leaves the line making students = [1,0] and sandwiches = [0,1].
	- Front student leaves the top sandwich and returns to the end of the line making students = [0,1].
	- Front student takes the top sandwich and leaves the line making students = [1] and sandwiches = [1].
	- Front student takes the top sandwich and leaves the line making students = [] and sandwiches = [].
	
	Hence all students are able to eat.

6	3

1 1 1 0 0 1 1 0 0 0 1 1	

8
1 1 0 0 0 1 1 0
0 1 1 0 1 0 0 1
	0

8
1 0 0 0 1 0 0 1
0 1 1 0 1 0 0 1
	1

8
1 0 0 0 1 0 0 0
0 1 1 0 1 0 0 1
	4

Task 08:

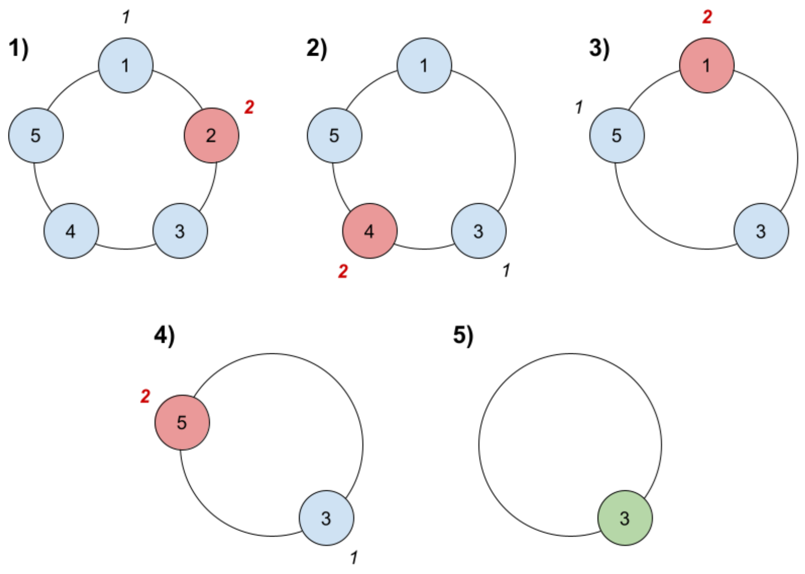
Description: There are n friends that are playing a game. The friends are sitting in a circle and are numbered from 1 to n in clockwise order. More formally, moving clockwise from the ith friend brings you to the (i+1)th friend for 1 <= i < n, and moving clockwise from the nth friend brings you to the 1st friend.

The rules of the game are as follows:

1. Start at the 1st friend.
2. Count the next k friends in the clockwise direction including the friend you started at. The counting wraps around the circle and may count some friends more than once.
3. The last friend you counted leaves the circle and loses the game.
4. If there is still more than one friend in the circle, go back to step 2 starting from the friend immediately clockwise of the friend who just lost and repeat.
5. Else, the last friend in the circle wins the game.

Given the number of friends, n, and an integer k, return the winner of the game.

Example 1:



Test case:

Input	**Output**	**Explanation**
n = 5, k = 2	3	Here are the steps of the game: 1) Start at friend 1. 2) Count 2 friends clockwise, which are friends 1 and 2. 3) Friend 2 leaves the circle. Next start is friend 3. 4) Count 2 friends clockwise, which are friends 3 and 4. 5) Friend 4 leaves the circle. Next start is friend 5. 6) Count 2 friends clockwise, which are friends 5 and 1. 7) Friend 1 leaves the circle. Next start is friend 3. 8) Count 2 friends clockwise, which are friends 3 and 5. 9) Friend 5 leaves the circle. Only friend 3 is left, so they are the winner.
n = 6, k = 5	1	The friends leave in this order: 5, 4, 6, 2, 3. The winner is friend 1.