CSE 4304-Data Structures Lab. Winter 2022-23
Date: 12 Sep 2023
Target Group: SWE B
Topic: Queue

Instructions:

- Regardless of when you finish the tasks in the lab, you have to submit the solutions in the Google Classroom. The deadline will always be at 11.59 PM of the day in which the lab has taken place.
- Task naming format: <fullID>_<Task><Lab><Group>.c/cpp. Example: 170041034_T01L02A.cpp
- If you find any issues in the 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 the <bits/stdc++.h> library.

Task-01:
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:
A queue is a data structure based on the principle of 'First In First Out' (FIFO). There are two ends; one end can be used only to insert an item and the other end to remove an item. A Double Ended Queue is a queue where you can insert an item in both sides as well as you can delete an item from either side.

There are mainly four operations available to a double ended queue. They are:

1. `pushLeft()` inserts an item to the left end of the queue with the exception that the queue is not full.
2. `pushRight()` inserts an item to the right end of the queue with the exception that the queue is not full.
3. `popLeft()` removes an item from the left end of the queue with the exception that the queue is not empty.
4. `popRight()` removes an item from the right end of the queue with the exception that the queue is not empty.

Now you are given a queue and a list of commands, you have to report the behavior of the queue.

Input

Input starts with an integer T (≤ 20), denoting the number of test cases.Each case starts with a line containing two integers n, m ($1 \le n \le 10$, $1 \le m \le 100$), where n denotes the size of the queue and m denotes the number of commands. Each of the next m lines contains a command which is one of:

Operation	Action
pushLeft x	pushes x (-100 ≤ x ≤ 100) to the left end of the queue
pushRight x	pushes x (-100 ≤ x ≤ 100) to the right end of the queue
popLeft	pops an item from the left end of the queue
popRight	pops an item from the right end of the queue

Output

For each case, print the case number in a line. Then for each operation, show its corresponding output as shown in the sample. Be careful about spelling.

Sample:

Input:

```
1
3 8
pushLeft 1
pushLeft 2
pushRight -1
pushRight 1
popLeft
popRight
popLeft
popRight
```

Output:

```
Case 1:
Pushed in left: 1
Pushed in left: 2
Pushed in right: -1
The queue is full
Popped from left: 2
Popped from right: -1
Popped from left: 1
The queue is empty
```

Task 03:
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.
```

<u>**Task 04**</u>

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>
#include <queue>

using namespace std;
```

```cpp
queue<int> q;
// 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.
```