Assignment #2, Part #2 of 2
Circular Queue Implementation

Instructor: Homeyra Pourmohammadali
BME 122 - Data Structures and Algorithms
Winter 2022
University of Waterloo

Due: 10:00 PM, Friday, February 18, 2022

Purpose of this assignment

In this assignment, you will practice your knowledge about **queue** by implementing a data type called **Circular Queue**. You need to use **circular array** to implement it. The header file `circular-queue.h`, which is explained below, provides the structure of the `CircularQueue` class with declarations of member functions. Do not modify the signatures of any of these functions. You need to implement all of the public member functions listed in `circular-queue.cpp`.

Instruction

Sign in to GitLab and verify that you have a project set up for your Assignment 2 (A2) at `https://git.uwaterloo.ca/bme122-1221/a2/WATIAM_ID` with the following files.

```
YOUR-WATIAM-ID
├── CMakeLists.txt
├── README.md
├── dynamic-stack.cpp
├── dynamic-stack.h
├── circular-queue.cpp
├── circular-queue.h
├── test.cpp
├── a2.1.cpp
└── a2.2.cpp
```

For this part of assignment, you only need to modify `circular-queue.cpp`. Do not change any part of the header file (`circular-queue.h`). You can design your own test case and code in `test.cpp`. It is optional and we will not grade this file.

You can use the same procedures in Assignment 0 to pull, edit, build, commit, and push your repo.

Description

The details of the header file `circular-queue.h` are as follows:

`QueueItem` defines the kind of data that the queue will contain. Being public, it can be accessed directly as `CircularQueue::QueueItem`.

`QueueItem EMPTY_QUEUE` defines a constant that will be used to indicate an empty queue. (Note that any actual data value stored in the queue should not be the same as this value.) Being public, it can be accessed directly as `CircularQueue::EMPTY_QUEUE`.

Member variables:
`items_`: An array of queue items.

`head_`: Index of the first element in the circular queue.

`tail_`: Index of the element after the last item in the circular queue.

`capacity_`: Maximum number of items in the queue.

`size_`: Current number of items in the queue.

Constructors and Destructor:
`CircularQueue()`: Default constructor of the class `CircularQueue`. It uses 16 as the capacity of the array and allocates the required memory space for the queue. The function appropriately initializes the fields of the created empty queue.

`CircularQueue(unsigned int capacity)`: Parametric constructor of the class `CircularQueue`. It allocates the required memory space for the queue of the given capacity. The function appropriately initializes the fields of the created empty queue.

`~CircularQueue()`: Destructor of the class `CircularQueue`. It deallocates the memory space allocated for the queue's items.

Constant member functions:
`int size() const`: Returns the number of items in the queue.

`bool empty() const`: Returns `true` if the queue is empty, and `false` otherwise.

`bool full() const`: Returns true if the queue is full, and false otherwise.

`void print() const`: Prints the queue items sequentially and in order, from the front to the rear of the queue. (Note: it is mainly used to help you visualize the data. It will not be used in any test cases for grading.)

`QueueItem peek() const`: Returns the value at the front of the queue without removing it from the queue. If the queue is empty, it returns the `EMPTY_QUEUE` constant instead.

Non-constant member functions:
`bool enqueue(QueueItem value)`: Takes as an argument a `QueueItem` value. If the queue is not at capacity, it inserts the value at the rear of the queue after the last item, and returns true. If the insertion fails due to lack of space, it returns false.

`QueueItem dequeue()`: Removes the item from the front of the queue and returns it. If the queue is empty, it returns the `EMPTY_QUEUE` constant instead.

Note

All indexes must start with 0.

Marking

We will try different inputs and check your output. We will only test your program with syntactically and semantically correct inputs.

Part 2 counts 50% of Assignment 2, which is 50 points in total.

Your program runs and does not crash during the test: + 10
Passes Test Cases: + 4 each, in total of 40