Lab 04: Queue with Dynamically-Allocated Array

Overview

This lab will practice queues using a dynamically-allocated array.

Queues

Queues are data structures that follow the FIFO (first in, first out) rule to control its enqueue and dequeue behavior. The enqueue operation adds new elements to the back of the queue. The dequeue function removes elements from the front of the queue. Queues have many real-life applications such as in networking (as seen in PA1).

Queues with Dynamic-Allocated Arrays

For PA1, you are tasked with coding a queue using a doubly linked list as its underlying structure, and in lecture, you saw an example of a circular queue in a predefined array of static size.

For today's lab, we are exploring the use of a dynamic array to achieve the same goal. Doing so changes the underlying operations needed to keep the high-level behavior of queues. Looking from outside of the `Queue` class, one should not see any change in the way it functions, no matter the underlying data-carrying structure.

Similarly to PA1, you are expected to implement the main queue functions using a template-focused solution that can adapt to any data type. For testing purposes, we include only the instantiation of the Queue with the integer type (`Queue<int>`) in the provided starter code.

Submission

You will need to implement four standard functions of the `Queue` data structure:

```
1. bool empty() const;
2. void enqueue(type elem);
3. void dequeue();
4. type* front() const;
```

The `empty` function returns `true` if the queue is empty or `false` otherwise.`Enqueue` takes an element as input parameter and adds such an element to the back of the queue. `Dequeue` removes the element at the front of the queue. The `front` function returns a pointer to the element at the front of the queue.

Hint: You need to reallocate the underlying array when enqueueing new values if the current capacity is not enough to hold all the elements. You should also be mindful of edge cases such as not allowing dequeue to throw an exception with an empty list.

Complete the "Lab 04. Queue with Dynamically-Allocated Array" activity on Mimir by uploading only the `Queue.cpp` source file. **You should not modify the provided header file. Any changes will not be considered by Mimir when testing your solution.**

You will implement four standard functions for the Queue data structure. A battery of automated tests will be run to ensure that the errors have been fixed and the results can be properly computed.

A grade of "complete" on this lab work requires a score of 100%.