A Customer Complaint Queue

Due 23:59 Monday, February 01, 2016

In this assignment you will design and implement a `CCQueue` class modeling a simple queue for an online computer equipment retailer's customer service department with many angry customers. The data storage of the `CCQueue` class will supported by a doubly-linked list template class.

Please review the general submission and coding style guidelines on the website.

Doubly-Linked List Description

Doubly-linked lists are dynamic reference structures much like the singly-linked lists seen in your lecture notes. Individual data elements are still stored within a node structure, although nodes in doubly-linked lists now contain both a pointer to the next list element as well as another pointer to the previous element in the list. The previous element pointer at the front of the list and the next element pointer at the back of the list are NULL. With such a doubly-linked structure, the list can be traversed towards the back by following the chain of next element pointers, and traversed towards the front by following the chain of previous element pointers.

A doubly-linked list can be visualized as follows:

You must implement the `DLinkedList` template class to store data of any type; this includes a Node template class implemented for you in the `DLinkedList` class .h file. Please refer to the documentation in the provided dlinkedlist.h file for the class definition and functional requirements.

ElementAt, InsertAt, RemoveAt example
Consider a linked list storing integers:

Front – 16 – 76 – 21 – 53 – back

Demonstrating 0-indexed access, `ElementAt(1)` returns 76. Likewise, `InsertAt(81, 2)` will result in the list, where 81 now occupies index 2:

Front – 16 – 76 – 81 – 21 – 53 – back

Subsequently, `RemoveAt(0)` returns 16 and results in the list:

Front – 76 – 81 – 21 – 53 – back

`CCQueue` class

The `CCQueue` contains a private `DLinkedList` member with a `Ticket` template type (provided in ticket.h and ticket.cpp). *The* `CCQueue` *public functions are to interact with the ticket queue using calls to* `DLinkedList` *methods only.* Please refer to ccqueue.h for the class definition and functional requirements.

Notes:
While the `CCQueue` `Service()` and `Add()` functions are based on some queue-like behaviours, the `MoveUp()`, `MoveDown()`, and `PrintStatus()` functions involve random access so `CCQueue` is not strictly a queue as discussed in class.

Error Handling

Your `DLinkedList` class is to throw exceptions on invalid inputs such as list indices out of bounds. `CCQueue` functions are to be restricted such that they will not call `DLinkedList` functions with any invalid inputs. See the comments in ccqueue.h for details on any exceptions that will be thrown by `CCQueue` functions.

Testing and Submission

As with assignment 1, a partial test driver has been provided for you. Note that while this driver will call every function you have been asked to implement, it is by no means a thorough test of each function's special cases and general cases. We will rigourously test both your `DLinkedList` and `CCQueue` classes separately; it is your responsibility to ensure that your classes function correctly for all general and corner cases of inputs.

Include the following deliverables in your submission as a ZIP archive titled assign-2.zip submitted to CourSys:

- Title page listing the information of all contributing group members
- dlinkedlist.cpp
- ccqueue.cpp