Customer Complaint Queue

Due 23:59 Saturday, May 28, 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.

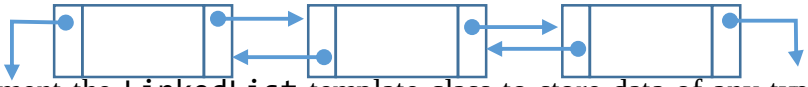
You are encouraged to work in groups of **at most** two members. You must include in your `cs221/a2` directory a `titlepage-a2.txt` file listing the names, student numbers, ugrad IDs, and lab sections of all contributing members. After using `make clean` to remove any compiled binaries and temporary files, use `handin cs221 a2` to submit your assignment. After you have submitted, you can verify that the handin was successful with `handin -c cs221 a2`. If you wish to overwrite your submission with a newer submission, use `handin -o cs221 a2`.

NOTE: electronic handin is expected to be configured by Wednesday, May 25. Please do not attempt to handin your submission before this date!

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 `LinkedList` template class to store data of any type; this includes a `Node` template class implemented for you in the `LinkedList` class .h file. Please refer to the documentation in the provided `linkedlist.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 `LinkedList` 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* `LinkedList` *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 `LinkedList` 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 `LinkedList` 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

A `Makefile` and partial test driver have 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 `LinkedList` and `CCQueue` classes separately; it is your responsibility to ensure that your classes function correctly for all general and corner cases of inputs.

Submit your assignment electronically from the ugrad machines using the commands described at the top of this document. Please ensure that your `cs221/a2` folder contains only the following files:

- `titlepage-a2.txt`
- `ticket.h`
- `ticket.cpp`
- `linkedlist.h`
- `linkedlist.cpp`
- `ccqueue.h`
- `ccqueue.cpp`
- `a2simpledriver.cpp`
- `Makefile`

Hints for this (and future) programming assignment

Submissions which do not compile will not be graded. It is recommended when you begin, to create your `.cpp` implementation files with stubs for all functions, i.e. they contain no logic other than to return a default value of the correct return type. An example function stub from `linkedlist.cpp` is shown below:

```
template <typename T>
bool LinkedList<T>::Contains(T item) const
{
  return false;
}
```

If you are unable to complete the functionality of any methods, simply leave them as stubs so that at least we will be able to compile and run tests for other functions.

For the collection classes such as `LinkedList`, many if not all of the functions can only be meaningfully tested if the collection first contains some items. Thus it is of utmost importance that the various `Insert` methods are fully tested and working before proceeding to implement other functions.

DRAW PICTURES! It will be invaluable in helping to visualising the sequence of steps that need to be performed for the various special cases and general cases of operations on your data structures. Make sure you are clear about each step-by-step process and conditions *before* you begin writing your code. Code which is written without a clear purpose is not likely to work as intended. Remember that computers are actually extremely good at following instructions, so make sure that you know what the right instructions are before you tell the computer what to do.

You are welcome to code and debug in any environment of your choice, however as we will be grading on the ugrad Unix system, it is highly recommended for you to test your program in the ugrad Unix environment using the supplied `Makefile` before making your submission.