COMP 2404 -- Tutorial #6

Linked Lists

Learning Outcomes

After this tutorial, you will be able to:

- replace one collection class with another, with no impact on other classes
- work more extensively with pointers by implementing a doubly linked list

Instructions

1. You will begin with the code you saved from Tutorial #5. You will need to be familiar with the linked list coding example from Section 4.1 (program #7) to do this tutorial.

2. Create a new `List` collection class, along with the corresponding `Node` class, that holds a **doubly linked list** of `Book` pointers. You will implement the linked list as we saw in class, with no dummy nodes. The `List` class will contain:
   - a data member for the head of the list, as a pointer to the first node in the list
   - a constructor
   - a destructor
   - an `add(Book*)` function that adds a new book to the list
     - the new book will be added in its correct position, in ascending order by year, using to the `Book` class's `lessThan()` function
   - a `print()` function that prints out the books to the screen
     - the output will contain all the book data from the list traversed in the forward direction, followed by all the book data from the list traversed in the backward direction, to prove that the double links in the list are correct

   Make sure that all dynamically allocated memory is explicitly deallocated when the list is no longer used. Where's a good place for this cleanup code?

3. Change the `Library` class to use a new `List` object instead of the existing `Array` object. There should be zero impact on existing classes because of this change, except for minor changes to the `Library` class. There should be no changes to the `Control`, `View`, and `Book` classes, nor to the `main()` function.

4. **[For bonus marks only]** Replace the `List` class's `print()` function with a `format(string& outStr)` function, as described in Tutorial #5. This function should format all the book data in the list into one big string. The data should first be formatted in the forward direction, then in the reverse direction.

5. Build and run the program. Check that the books are ordered correctly, both in the forward direction and the backward direction, when the library is printed out at the end of the program.

6. Make sure that all dynamically allocated memory is explicitly deallocated when it is no longer used. Use `valgrind` to check for memory leaks.

7. Package together the tutorial code into a tar file, and upload it into cuLearn. Save your work to a permanent location, like a memory stick or your Z-drive.