CSci 115 Spring 2016
Lab 4
(due Tue Mar 1, 6:00pm)

For this lab assignment you will be providing an implementation of the ADT given in the `List.h` header file on Piazza, which specifies a two-ended list with bi-directional navigation and (free-list-style) memory management. Specifically, you will need to:

- Create a `Link` template class for storing a single node of a doubly-linked list; this `Link` class should have
  - three **static** private member variables to keep track of the free list, the number of active nodes, and the number of free nodes. Because these are static, they are "global to the class": every instance of the class will share these variables (not have their own copies)
  - one or more constructors for the class
  - an overloaded `new` operator that grabs a `Link` from the free list if it is not empty and otherwise allocates a new one, and an overloaded `delete` operator that puts the `Link` on the beginning of the free list; these should be written in such a way that they correctly keep track of (in the static variables) the total number of active nodes and the total number of free nodes.
- Create a `DLList` template class that inherits from the abstract `List` template class (in `List.h`) and uses `Link` to implement its virtual methods. These operations should have the following features:
  - The `clear()` operation should free (delete) all the Links in the list
  - `prepend()` and `append()` should allocate new nodes (which, of course, because of the overloaded `new` operator, should grab them from the free list, if possible)
  - `numActive()` and `numFree()` should return the values of the associated static variables of a `Link` object (any one of them, since they all share these variables)
  - each non-empty list has a current node, which starts out being the first element of the list, and is advanced and backed-up using the `next()` and `prev()` methods, and reset using the `moveToStart()` and `moveToEnd()` methods; the `getValue()` method returns a pointer to the current node, if it exists
  - lists should only contain nodes with data; i.e., there should be no header nodes at the beginning or end of the list.

These two class definitions will go into a `lists.cpp` file that `#include`s the abstract class header file (`List.h`).

In addition, you will need to create a `driver.cpp` file that also `#include`s the abstract class header file, and then runs a series of tests that exercises all of the features of your linked list implementation. The driver should print out information about which of your tests have passed and which have failed. **You will be graded in part based on how complete your testing is.**

Turning in your lab

As usual, clear out your `turn-in` directory and place there, before the 6:00pm deadline on March 1, the two files lists.cpp and driver.cpp.