

Due: Thursday, June 23

Emphasis: Linked Lists, Creating a Class Library

In this assignment we will be fleshing out the books description of a pointer-based implementation of a linked list class (section 6.4, p. 295-300). In the assignment you are to build a complete **List** class, using a pointer-based linked-list implementation as described in the section 6.4. For basic operations your **List** class should have a constructor, destructor (since you will be using dynamically allocated memory), and the basic list operations: empty, traverse, insert, and delete.

You will be required to implement your **List** class using the standard convention of splitting up the definition into a header file (called List.h) and the implementation of your class methods into an implementation file (List.cpp). I have given you a header file, List.h, with most of the declarations you are required to implement. You will need to implement all of the member functions and use the data members as they are declared in the given List.h header file. You may or may not find it helpful to add other member functions and/or variables while implementing you list class. Therefore you may need to make changes to List.h to add other things you need for your implementation.

In addition to you List class, you should create a small test program (in a third, separate file) to demonstrate your implementation of your linked **List** class. The test driver should create a List object, using your class constructor, demonstrate the empty operator, insert and delete some items and demonstrate the traversal of the items in the list.

Requirements

- 1) You must implement your **List** class by fleshing-out the template given for the pointer-based implementation on page 296.
- 2) Use the private Node class, pointer-based implementation to represent the nodes of your list.
- 3) Use dynamic memory allocation in order to create new nodes to add items when they are inserted into your **List**. Be sure to correctly free the memory when items are deleted from the list, and when the list is destroyed (see #5 implementing a destructor).
- 4) Implement a constructor that initializes the **List** to be initially empty. Extra credit will be given for implementing a copy constructor for you **List** class.
- 5) Implement a class destructor to free up all of your classes' nodes when the class is destroyed.
- 6) Your *empty* member function should return a boolean value that is true if the list is empty (e.g. contains no items) and is false when the list has 1 or more items in it.
- 7) The *traverse* member function should simply display all of the items in the **List** in readable format. As shown in the **Time** classes' *display* method from chapter 4, implement traverse as an output function that receives an ostream as its only parameter and displays the list elements out to the given ostream.
- 8) *Insert* should receive an ElementType and insert it into the **List**. You can insert new items at the head of the **List** as one easy way of implementing insert.
- 9) *Delete* should also receive an ElementType, then search the **List** for items equal to the indicated ElementType and remove them from the **List**. You can either simply remove the first such item in your **List**, or remove all equal items.

The *insert* and *delete* requirements in 8 and 9 are slightly different than the algorithm described for *insert* and *delete* in our book in sections 6.1 & 6.2. You should find it easier to implement *insert* and

delete as described in 8 and 9 above, but if you prefer you may try and implement your *insert* and *delete* as described for the **List** abstract data type in chapter 6 of our text.