CSC136 – Program 4, LinkedList

Assignment: Complete and test a template class implementing storage and manipulation of a linked list.

Supplied: You will be provided, as a starting point, with incomplete versions of classes named **LinkedList** and **Employee,** as well as a completed **LListIter** iterator class. You are also given **etest.cpp**, a stand-alone test for the Employee class and a partially complete test program called **LLtest.cpp** as well as a file containing test data named **data**. Complied and runnable solution executables for both of these are provided for comparison and are called **esoln** and **llsoln**, respectively***.***

Location: **/export/home/public/carelli/csc136/Projects/Project4**

Deliverable: You should turn in the completed classes and test program (see Grading Critera). Specifically, turn in *only* **LinkedList.h**, **Employee.cpp**, and **LLtest.cpp**

Due: The program MUST be turned in by the assigned date and time using the ***turnin*** script. Late submissions will, in fact, be rejected by the script, resulting in a grade of zero.

Overview:

For this assignment you will be given a partially complete implementation of a linked list class in a file called **LinkedList.h**. In addition, a file called **ListIter.h** contains an iterator class, which gets included, and used, in **LinkedList.h**. The iterator is complete, so no work needs to be done on it. The assignment directory also contains a partially complete driver program, called **LLtest.cpp**, to test the linked list implementation. A working executable called **llsoln** produces the output that the program should produce. There is also an implementation of a class called **Employee**, in two files, which is used in the test program as well as a **makefile** for compilation. A stand-alone test program for the Employee class, together with a working executable called **etest.cpp** and **esoln**, respectively, are provided as well.

Employee:

The first part of the assignment is to complete the Employee class. There is one method, getTotal() and three comparison operator overloads that need to be completed, as follows:

- getTotal() return the total sales for the employee
- operator==() do the employees have the same name?
- operator>=() alphabetic comparison of the employee names
- operator<=() numerical comparison of the employee total sales

Incomplete versions of these will be found at the end of the **Employee.cpp** file. Once completed, the results produced by the compiled stand-alone **Employee** test program, **etest.cpp** should agree with what is produced by the provided solution executable **esoln**.

LinkedList:

Write the code needed to implement the **LinkedList** methods, described below, which are missing or incomplete. Empty implementations for the incomplete methods are at the end of the **LinkedList.h** file as indicated by comments in the file. They are the following:

- void add(T) – add an item to the end of the linked list.
- bool replace(T)– replace an existing item that matches the supplied item with the supplied item returning true if successful, false otherwise
- Node<T> *extract(T) – extract the item with the given value from the list. *Note: the extracted node is **NOT** deleted. Instead, its pointer is returned. If the value is not in the list, the NULL pointer is returned. Note also that this is a private, helper method used for sorting (see below).* ***Hint: Refer to the bool remove(T) method – it performs a similar function!***
- T findMax() – return a copy of the maximum value in the list. If the list is empty, return the default value for the data type T.
- void reverse() – reverse the order of items in the list *using pointer manipulations*.

The *findMax()* and *extract()* methods are used by an already complete sorting method called *sortAscend()*, which will sort the elements in the list in ascending order.

One more important note! extract() and reverse() should be implemented purely with pointer manipulations. Do not create or delete any nodes.

Finally, in addition to the methods listed above, you will also need to add the following methods:

- T findMin() – same as findMax(), but for the minimum value in the list.
- sortDescend() – same as sortAscend() (already written), but sorts in descending order.

LLtest:

LLtest.cpp, from which the **makefile** will produce the executable **lltest**, is a command-based driver program. It uses a linked list of **Employee** objects. After reading a file with employee data, it provides a list of available commands that can be executed. It will be demonstrated in class - **llsoln** is a working version of the program that is also provided for comparison.

As the methods and operators detailed above are completed, different commands will begin to work. In addition, you will need to complete several of the commands yourself, using the provided commands as examples.

Part of the job is to examine the code and understand how the objects interact and make use of defined operators and methods to accomplish the targeted behavior.

Major dependencies are as follows:

Commands:

readFile:	LinkedList add() (when the program starts)
Find an Employee:	Employee ==
Total Sales:	Employee getSales()
Name sort:	LinkedList sortAscend(), extract(), and findMax()
Add a Sale:	LinkedList replace()

In addition to these, you will need to complete the code to implement the following commands:

Hire a new employee:	LinkedList add()
Sales sort:	LinkedList sortDescend(), extract(), and findMin()
Reverse list:	LinkedList reverse()

It is recommended that the methods be completed in the order listed - do the add() method first. As they are completed correctly, the output of **LLtest** will begin to match what is provided by **LLtestSoln**. The *add()* and *findItem()* methods will enable proper operation of the initial tests. *findMax()* and *remove()* will enable Name sorting to function correctly. Add the sales sort yourself. The last test depends on *reverse()*. When they are all complete, the results for the integer linked list should match.

Grading Criteria:

1. Turn in *only* **LinkedList.h**, **Employee.cpp**, and **LLtest.cpp** on time using turnin – late programs will receive a penalty. Very late programs will not be accepted. (10%)
2. Program must compile. (10%)
3. The program must be well commented/documented. (10%)
    a. Using the departmental standard for documentation.
    b. Include comments within your code to highlight important operations/techniques
    c. Note: you only need to add departmental standard documentation for the functions/methods that you write or modify – don't worry about the rest.
4. Properly implement the given **Employee** methods/operators. Test with **etest.cpp**. (15%)
5. Properly implement the **LinkedList** methods. (35%)
    a. The methods given and…
    b. The additional methods listed above
    c. Follow the directions given
    for ex. - only use pointer manipulations in extract() and reverse()
6. Properly complete the **LLtest** code for the missing commands (given above). (20%)

Templates for adhering to the departmental documentation guidelines can be found in ***CodeDocTemplate.txt*** under the Documents link on the course webpage.