comments on feedback (thanks again!)
when should we use private and public?
friend functions
destructors
why operator overloads?

CptS 122 – Data Structures

Lab 7: Developing a (Linked) List Class in C++

Assigned: Week of February 24, 2025
Due: At the end of the lab session

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Design, implement, and test a List class in C++
- Apply a customized linked List class
- Compare and contrast shallow versus deep copy
- Compare and contrast value classes versus container classes
- Apply and implement *overloaded* functions and operators

II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Analyze a basic set of requirements for a problem
- Create test cases for a program
- Design, implement and test classes in C++
- Declare and define *constructors*
- Declare and define *destructors*
- Compare and contrast *public* and *private* access specifiers in C++
- Describe what is an *attribute* or data member of a class
- Describe what is a *method* of a class
- Apply and implement *overloaded* functions
- Distinguish between pass-by-*value* and pass-by-*reference*
- Discuss *classes* versus *objects*

III. Overview & Requirements:

This lab, along with your TA, will help you navigate through designing, implementing, testing a List class in C++. It will also help you with understanding how to apply a List object to an application.

Labs are held in a "closed" environment such that you may ask your TA questions. Please use your TAs knowledge to your advantage. You are required to move at the pace set forth by your TA. Have a great time! Labs are a vital part to your education in CptS 122 so work diligently.

--- Read the following before you start! ---
A couple of notes to consider:

- You will be working with *dynamically* allocated space in C++. Instead of using `malloc()`, we'll use operator `new` to allocate space. Also, instead of using `free()` to deallocate space, we'll use operator `delete`. When operator `new` is called, the constructor for the object is automatically invoked! When the operator `delete` is called, the destructor for the object is automatically invoked! Please keep these ideas in mind.

- You will be working with *copy* constructors. You will have the option to implement a *shallow* and *deep* copy constructors. A *shallow* copy will perform a basic assignment of data members of one object to the same data members of another object. A *deep* copy will allocate extra space to ensure that the items copied are in distinct locations. A deep copy is usually considered when working with pointers! If you want to copy one list to another list, should the head pointer be copied only (shallow copy)? Or should extra space be allocated to establish a completely new list with the same data items (deep copy)?

Tasks:

Starting with the List code found at https://eecs.wsu.edu/~aofallon/cpts122/labs/ClassLinkedList.zip:

1. Unzip the file and review "main.cpp" with your teammates. Answer questions 1-6, which are provided through comments in the file. Use the debugger to help answer the questions!

2. Review the "ListNode.h", "ListNode.cpp", "List.h", and "List.cpp" files. Start to fill in the code for the functions listed below. Note: the ListNode class declaration ("ListNode.h") and ListNode function implementations ("ListNode.cpp") have been completed for you!

   a. Write the implementation for the deep copy constructor the List class (found in "List.cpp"). After you finish writing your copy constructor, uncomment line 25 (`List l2 = l1;`) in main (). Answer questions 7 and 8.
   b. Write the implementation for the deep copy assignment operator (found in "List.cpp"). After you finish writing your overloaded assignment operator, uncomment line 37 (`l3 = l2;`) in main (). Answer questions 9 and 10.
   c. Write the implementation for insertInorder () (found in "List.cpp"). Test your function by calling it in main () or within a test function.
   d. Write the implementation for insertAtEnd () (found in "List.cpp"). Test your function by calling it in main () or within a test function.
   e. Write the implementation for deleteAtFront () (found in "List.cpp"). Test your function by calling it in main () or within a test function.
   f. Write the implementation for deleteNode () (found in "List.cpp"). Test your function by calling it in main () or within a test function.
   g. Write the implementation for deleteAtEnd () (found in "List.cpp"). Test your function by calling it in main () or within a test function.
   h. Answer question 11 in main ().

3. Review the "ListApp.h" and "ListApp.cpp" files. Write an application, which computes the high, low, and mean of the scores in the "ExamScores.csv" file. The functions for reading and extracting the scores from the file, and for inserting into the list have been completed for you! See the "ListApp.h" and "ListApp.cpp" files. You will need to write functions for the computations and for writing the results to a file. You will have to visit each node in your linked list of scores to perform the computations! The high, low, and mean should be written to a file called "ExamStats.txt". These functions should be called from the runApp () function. Note: the ListApp class constructor takes care of opening the files!

IV. Submitting Labs:

- You are not required to submit your lab solutions, unless you are unable to attend them synchronously. You should keep them in a folder that you may continue to access throughout the semester.

V. Grading Guidelines:

- This lab is worth 10 points. Your lab grade is assigned based on completeness and effort. To receive full credit for the lab you must show up on time, work with a team, complete 2/3 of the problems, and continue to work on the problems until the TA has dismissed you.