CSc 212 Data Structures

## Programming Project 4 (Chapter 5) - Revised sequence class with the Linked List Toolkit

**The Assignment:**

You will implement and test a revised sequence class that uses a linked list to store the items.

**Purposes:**

Ensure that you can write a small class that uses the linked list toolkit to create and manipulate a linked list.

**Before Starting:**

Read all of Chapter 5, with particular attention to Sections 5.3 and 5.4.

**Due Date:**

Monday, July 1st before class

**How to Turn In:**

Upload the sequence3.h, sequence3.cpp, node1.h, node1.cpp and screenshot of the output

of sequence\_exam3.cpp to Canvas.

**Files that you must write and turn in:**

1. sequence3.h: The header file for the new sequence class. Actually, you don't have to write much of this file. Just start with our version and add your name and other information at the top. If some of your member functions are implemented as inline functions, then you may put those implementations in this file too. By the way, you might want to compare this header file with your first sequence header file (sequence1.h), and second sequence header file (sequence2.h). The linked list version no longer has a CAPACITY constant nor a DEFAULT\_CAPACITY constant because the items are stored on a linked list instead of an array.
2. sequence3.cpp: The implementation file for the new sequence class. You will write all of this file, which will have the invariant of the sequence class, the implementations of all the member functions of the sequence class, and the Big-Os of these functions.
3. node1.h and node1.cpp: Copy these files. They contain the linked list toolkit from Section 5.2. You may use these files without changing them. However we only provide the documentation for the function list\_piece (see Self-Test Exercise 24 on page 258). You may need to write an implementation of this function if you are going to use it in writing your copy constructor and overloading your assignment operator. If you do make changes, please turn them in.

**Other files that you may find helpful:**

1. sequence\_test.cpp: This is the same interactive test program that you used with the earlier sequences. This is exactly what we expect for the information hiding of the class - the users do not need to know how the class is implemented (both the member variables and the functions) as far as the interfaces (the prototype of the member functions) remain the same. So if you want to use the test program with the new sequence, the only thing you need to do is to copy it to your directory and open it with your editor. Then change the statements

#include "sequence1.h"   
using namespace main\_savitch\_3;   
to   
#include "sequence3.h"   
using namespace main\_savitch\_5;

1. sequence\_exam3.cpp: A non-interactive test program that will be used to grade the correctness of your new sequence class. The points given by the program will only be served as a reference; we are going to look into your code and your analysis (in the form of comment lines) for actual grading.

### The sequence Class Using a Linked List Discussion of the Assignment

Your sequence class for this assignment will differ from your previous sequence in the following ways:

* The sequence's items are now stored on a linked list. The head pointer of the linked list is a private member variable of the sequence class. I suggest that you also have a tail pointer as an additional private member variable of the sequence class. The reason for the tail pointer is explained in Section 5.4 of the textbook.
* Because you are dynamically allocation memory within your sequence class, you will need to define a copy constructor, an assignment operator, and a destructor. You need to pay special attention to the value semantics of your new sequence class - you need not only to make a copy of the linked list, but also need to place the node pointers correctly. Please refer to page 276 for more detailed discussions on the value semantics.

Start by declaring the new sequence's private member variables in sequence3.h. You might try declaring these variables yourself, and then compare your solution with the suggestion in Section 5.4.

Once again, do your work in small pieces. For example, my first version of the sequence had only a constructor, start, insert, advance, and current. My other member functions started out as stubs.

Design a sequence class using a linked list could be little bit more complicated than using an array. Therefore, I recommend you to draw an example linked list such as the one shown in page 267 of the textbook when you are writing code for each member function. Always remember to show all the member variables in order to remind you making proper changes of them. You need to place the cursor and the precursor at various locations, head, tail and other places. Note that the sequence could be empty, and cursor and/or precursor could be NULL, even when the sequence is NOT empty. With the help of small drawings, jobs will be much easier!

Use the interactive test program and the debugger to track down errors in your implementation. If you have an error, *do not start making changes until you have identified the cause of the error.*

**Run Time Analysis of the sequence classes and grading rules**

We will use the number of items in a sequence as the input size n.  Please give the Big-O of each function in your implementation, and compare them with the corresponding functions of the sequence using a dynamic array. You should write the time analysis in the comment lines of each function.  The breakdowns of points (of 100) will be the followings

Basis points (70) if your implementation passes the seq\_ex3 test   
Invariant of the class (5 points)   
Run time analysis (10 points)   
Other implementation details (15 points)