

# CST8130: Data Structures

## Lab 4: Linked Lists

### Introduction

In the preceding labs, you've worked with earthquake data. The data set consists of over 150,000 records, but in many cases, several records relate to a single earthquake event. There really aren't 150,000 earthquakes. In essence, multiple records for a single event provide different perspectives from the cities where that earthquake was experienced.

Ultimately, better organization would have all event-specific records clustered together. In this lab, you'll work with a single earthquake event, and all its related perspectives. (I've provided a sample data file that distills the 18 records that pertain to a single event.)

Your task will be to implement a new data structure – a linked list (eliminating the array of *EarthQuakeRecord* objects and the arrays of pointers to those objects). All of the event records will be loaded into a single linked list.

### Details: Implement a Simple Linked List

Your linked list can be the simplest form, that is, a list with only a head pointer (no tail) with each list node having a pointer to the next item.

Your *List* class will need several functions:

- *List*: Constructor to initialize variables that identify that the list is empty.
- *~List*: Destructor that ensures that dependent memory is released when the list object is disappearing. (The destructor can simply call the *ReleaseAllNodes()* function.) Note: You should consider implementing the Microsoft code that monitors memory leaks.
- *AddToHead*: Creates a new list node each time an additional *EarthQuakeRecord* object is to be stored.
- *Display*: Displays all *EarthQuakeRecord* objects in the list.
- *ReleaseAllNodes*: Removes all *ListNode* objects and sets the *List* variables to show that the list is empty.

The *ListNode* class must be able to manage a single *EarthQuakeRecord* object.<sup>1</sup>

You'll need a *Read()* function to load and parse the data from the file. You'll need a *Display()* function.

### Detailed Steps: Memory Map

You need real clarity in your memory map to convince me that you understand the code you're building. You will need to decide how the *ListNode* object will manage the *EarthQuakeRecord* (note the previous footnote).

### Submission Requirements

You will need to include the following:

- Problem Statement.
- Test Plan that validates the correctness of your work. Here, your test plan should clarify how you intend to use the debugger to inspect memory-resident data. Later you'll use the display function to view data. You may also need to set breakpoints at strategic locations.
- Source Code (with extensive comments that convince me that you understand your work).
- Memory Maps to clarify how memory will be organized based on your implementation.

Note: No PDL is required for this lab. The key to understanding this work is your memory map, and your casting and dereferencing operations.

---

<sup>1</sup> This can be done in one of two different ways: let the *ListNode* object have an *EarthQuakeRecord* stored inside; or let the *ListNode* object have a pointer to an *EarthQuakeRecord* object and leave the *EarthQuakeRecord* object external to the *ListNode*.