

# CS 216

## Laboratory 1:

### Linked Lists

*Objective:*

This laboratory introduces you to some advanced class development in C++, including the using template-classes for collections, creating and using iterators, manipulating pointers and linked data structures. It also addresses some issues involving testing and software development.

*Background:*

The linked list is a basic data structure from which one can implement stacks, queues, sets, and many other data structures. Lists may be singly- or doubly-linked. In this lab we will implement a doubly-linked list.

*Post-lab due:*

11:59 pm Thursday September 8, 2005

## Code Description

Linked lists are described in Chapter 3 of Weiss' book *Data Structures and Algorithm Analysis in C++*. Doubly linked lists are described on p. 79.

The code in the book implements singly linked lists using a dummy header node. ***You should implement your doubly linked list with dummy nodes as well*** - although you probably would want two - one for the head and one for the tail. One of the benefits of doing your implementation using dummy nodes is that there are fewer special cases to check for - for example you never have to update the head pointer on an `insert_before` or a `remove` - the head pointer always points to the dummy header node! A dummy tail pointer would help out in the same respect. The downside is: you use two extra "empty" nodes. The book describes these issues on page 72.

---

For this lab you will need to implement (at least) three classes: `ListNode`, `List`, and `ListIter`. For simplicity we will just create a list that holds integers (your code could easily later be templated to allow it to contain objects of other types). ***Please use the method names listed below in your code.***

### **ListNode**

A `ListNode` contains an integer value, as well as "next" and "prev" pointers to other `ListNode`s.

### **List**

This class represents the list data structure containing `ListNode`s. It has a pointer to the first(head) and last(tail) `ListNode`s of the list, as well as a count of the number of `ListNode`s in the `List`. Your `List` class should implement at least the following public methods:

<code>boolean isEmpty()</code>	Return true if empty; else false
<code>void makeEmpty()</code>	Remove all items
<code>ListIter first()</code>	Return an iterator that points to the <code>ListNode</code> in the first position
<code>void insert_after( int x, ListIter p )</code>	Insert x after current iterator position p
<code>void insert_before( int x, ListIter p )</code>	Insert x before current iterator position p
<code>void insert_at_tail(int x )</code>	Insert x at tail of list
<code>void remove( int x )</code>	Remove the first occurrence of x
<code>ListIter find( int x )</code>	Return an iterator that points to the first occurrence of x
<code>int cardinality()</code>	Returns the number of elements in the list.

In addition, you must implement these functions:

- a destructor, copy constructor, and `operator=` for the `List` class.

- a **non-member** function `printList()` that prints a list either forwards (by default – from head to tail) or backwards (from tail to head). *You must use your `ListIter` class to implement this function.*

### **ListIter**

Your `ListIter` should maintain a pointer to a “current” position in a List. An example of an iterator for a singly linked list can be found on p.74 of Weiss. Your iterator class will probably look a bit different from this. Your `ListIter` class should implement at least the following public methods:

<code>boolean isPastEnd()</code>	Return true if past end position in list, else false
<code>boolean isPastBeginning()</code>	Return true if past first position in list, else false
<code>void move_forward()</code>	Advance current to the next position in the list (unless already past the end of the list)
<code>void move_backward()</code>	Move current back to the previous position in the list (unless already past the beginning of the list)
<code>int retrieve()</code>	Return the item in the current position

### **Test Harness**

We have posted a test harness on the labs page. *The classes you implement must work with this test harness.*

## Lab Procedure

### Pre-lab

0. Read Weiss' *Data Structures* book pp. 69-81 on linked lists.
1. Implement the three classes as described on the previous page. You should have most if not all of the code working *before* coming to lab. TAs will be available to help in lab if you still have questions.
2. Create a list of test cases to use to test your classes. The TAs may ask you to test more cases during lab.
3. Read through the remainder of this document before coming to lab.
4. Each student needs to bring a copy of the checkoff sheet with them to lab.

### In-lab

*Come to lab with a printed copy of the checkoff sheet.*

0. Verify to yourself that your methods are working properly.
1. Fill in the checkoff sheet information *before* calling over a TA.
2. Demonstrate the functionality of your classes to one of the TAs.
3. Carry out the lab exercise on how to use the Visual Studio debugger (including how to follow pointers using the debugger).
4. Be sure to LEAVE your checkoff sheet with the head TA before leaving lab.

### Post-lab

0. For this lab you will be submitting your code electronically via the toolkit. Your fully functional code (3 classes) (submit the test harness – ONLY IF YOU MODIFIED IT FOR ANY REASON) is **due at 11:59 pm Thursday September 8, 2005**. Be sure to include: your name, the date, and the name of the file in a banner comment at the beginning of each file you submit. A link for turn-in will be posted on the labs page before that time.

**\*\*\* To Be Turned in BEFORE leaving lab.\*\*\***

## CS 216: Program and Data Representation Checkoff

<b>Name:</b>	<b>Circle:</b> <b>1 Tue 7:15 – 9:15 PM</b> <b>Your</b> <b>2 Tue 12:15 – 2:00 PM</b> <b>Section:</b>
<b>Date:</b>	<b>Email:</b>

<b><u>Lab 1: Linked Lists</u></b>	
<b>Demonstrated insert_before()</b>	
<b>Demonstrated insert_after()</b>	
<b>Demonstrated insert_at_tail()</b>	
<b>Demonstrated remove()</b>	
<b>Demonstrated cardinality()</b>	
<b>Demonstrated ListItr class</b>	
<b>Completed debugging exercise</b>	

TA comments on any of the above (TA should initial comments as well):

The work being evaluated is my own work: \_\_\_\_\_  
(student signature)