Lab 6: Dynamic Memory and Linked Lists

# Overview

In this lab you will:

* Learn how to use C++ dynamic memory operators
* Learn about the "linked list" and "queue" data structures

Basic Structure

In this lab you will implement a linked list to practice using dynamic memory. However, you will only implement a small subset of the functions that a full linked list would have – enough to implement what is known as a *queue*. A queue only supports adding new items to the end of the list (the *tail*), and extracting items in order from the start of the list (the *head*).

For an introduction to linked lists, see Chapter 17 of the textbook or this [tutorial](http://www.cs.cmu.edu/~adamchik/15-121/lectures/Linked%20Lists/linked%20lists.html) from Carnegie Mellon University.

Your assignment is to finish implementing three of the member functions of the classList: AddToEnd(), RemoveFromStart(), and Size().

The List class implements a linked list of integers. It consists of an object of the classList, which has a pointer to the first Node. Each Node in turn has a pointer to the nextNode in the list. There is a dummy Node, that is always present as the first item in the list, i.e., the List object's m\_head points to a dummy Node. This node contains no real data (the m\_data field is just set to 0), and is not considered to be an element of the list. It is there because it makes the implementation much easier: you can count on even an empty queue having at least one Node.

Getting the Files

If you've ssh’ed into the GL servers, issue the following commands at the command prompt while you’re inside your lab6 directory:

cp /afs/umbc.edu/users/c/m/cmarron/pub/cmsc202/lab6.zip .

unzip lab6.zip

(Note the '.' at the end of the command — that is very important.)

You should now have the following files in your working directory:

1. [List.h:](http://www.csee.umbc.edu/courses/undergraduate/202/fall15_marron/labs/lab06/List.h) the header file for the List class. You will not need to change this file.
2. [List.cpp:](http://www.csee.umbc.edu/courses/undergraduate/202/fall15_marron/labs/lab06/List.cpp) the implementation file for the List class. Most of the member functions have been implemented for you. Some parts of the functions have been omitted.
3. [lab6.cpp:](http://www.csee.umbc.edu/courses/undergraduate/202/fall15_marron/labs/lab06/lab6.cpp) a file with a main function that exercises the List class. You will not need to change this file.

Complete the Size() function

The Size() function should return the number of elements in the list, not including the dummy node.

Follow these steps:

1. Edit List.cpp. You will edit List::Size() near the bottom of the file.
2. Write a while loop that traverses the linked list with a current pointer, as in other class functions. Keep track of how many nodes have been visited with a counter, then return that counter.

Complete the AddToEnd() function

Insert a node that stores data at the end of the list. The traversal of the linked list to get to the end has already been written for you.

1. Edit List.cpp. You will edit List::AddToEnd(int data).
2. Implement the pointer logic in the section of code labeled "PUT CODE HERE". For example, suppose the linked list is
3. m\_head -> 1 -> 4 -> 5 -> 7 -> 8 -> NULL

and you want to add 6. current would already be pointing to the 8: your job is to allocate a new Node from the heap, initialize it, and insert it into the list after the 8 so that the list looks like

m\_head -> 1 -> 4 -> 5 -> 7 -> 8 -> 6 -> NULL

Complete the RemoveFromStart() function

Remove the first real data node in the list (skipping the dummy node), and return the value stored in it.

1. Edit List.cpp. You will edit List::RemoveFromStart(int data).
2. Implement the pointer logic in the section of code labeled "PUT CODE HERE". For example, if the linked list is:
3. m\_head -> 1 -> 4 -> 5 -> 7 -> 8 -> NULL

You would delete the "1" at the head of the list, returning the value to the caller. You must update the linked list so that that node is no longer linked into the list. You must also remember to free up the Node object itself by calling delete. The resulting list would be:

m\_head -> 4 -> 5 -> 7 -> 8 -> NULL

# Compiling and Testing

g++ -Wall -ansi lab6.cpp List.cpp -o lab6

## Run

linux3[22]% ./lab6