# Programming Assignment 1: Linked Lists

Due:

Monday Feb 12, by 8:00AM

Key Concepts:

Linked-Lists

Memory allocation and deallocation in C++

Runtime (a little)

Recursion

**Summary**: you have been given a C++ implementation of a list ADT in which some functions are already written and some are not. Your job is to complete the unwritten functions. (In one case you will make modifications to improve the runtime of one already implemented function).

**Instructions**: Examine the file List.h. There you will find an implementation of a List class with some functions not yet implemented (you will find "stubs" for these functions.

If you search for the string TODO you will find all of the requirements. In most cases each task is the implementation of a particular member function. The one exception is the length function which already exists, but you need to make modifications so that it runs in constant time -- these modifications are mostly outside of the length function itself).

**Your Job:** completion of all of the TODO items specified -- associated comments give details on the expectations.

**Rules**:

**YOU CANNOT:**

**Modify any of the function "signatures" (parameters and return types)**

**Introduce any global variables.**

**YOU CAN:**

**Add helper functions; you should make them private. Note: if you want helper functions that do not have a "calling object" (like in traditional C), you designate them as static.**

**Add data fields to the List class (to achieve one of the runtime requirements, a "bookkeeping" field will come in handy!).**

**Discussion**: The given linked list structure has two “levels”:

Here is a diagram of a list with three entries: <3, 8, 2>. The struct at the left (a LIST) gives access to the actual nodes.



There are 15 functions tagged with the word TODO in List.h. The points for each function is given in the table below.

Detailed descriptions of the requirements of each function are given in banner comments above the functions themselves.

(Note that the unwritten functions already have placeholder "stubs" for them (i.e., the functions technically already exist, but their bodies are empty).

|  |  |
| --- | --- |
| Function | Points |
| equal\_to | 10 |
| count | 10 |
| length | 15 |
| pop\_back | 15 |
| print\_rev | 15 |
| insert\_sorted | 15 |
| concat | 15 |
| clone | 15 |
| reverse | 20 |
| prefix | 20 |
| filter\_leq | 20 |
| merge\_with | 20 |
| fast\_remove\_all | 20 |
| compare\_with | 15 |
| suffix\_maxes | 15 |

The grand total is 240 possible points.

**Recommendations**

Start with the easier functions.

Do lots of testing! Write multiple driver programs which perform lots of stress tests on the List class. Think carefully about boundary cases, etc.!

Develop things "incrementally" -- after you've written a function, test it extensively before moving on.

Think carefully about "invariants" -- for example, "after I've completed some operation on a List have I made sure that front and back are correct in all possible scenarios?"

Look out for memory leaks!

**Submission Details:**

Submission will be via Blackboard. Details forthcoming.