Linked List lab (only head pointer) – Day 2

1. Get dropbox code. Read through all of the starter code.
   Things written already: constructor, print, prepend, push_back. Make sure you understand this code!

2. Write remove_head(). This function removes the node at the head of the list. No traversal is needed. There are two cases – list is empty and list is not empty.

3. Write remove_data(). This searches for and removes an item in the linked list by its data value, as opposed to always removing the head. You will need to traverse the list.

   Hints: Break this into 5 steps:
   a. Check if the list is empty, handle this case separately.
   b. Check if the head of the list contains our item; handle this separately.
   c. If we get here, our item must not be the head, so traverse the list to find it. Use the before/curr traversal technique to stop when we reach the item we are looking for.
   d. Did we walk off the end of the list?
   e. If not, remove the item we want to remove.

4. Write the destructor. The destructor should traverse the list and delete all the nodes within. This is tricky! Don't delete a node if you plan on accessing its next pointer immediately after! You will need to use the before/curr traversal idea.

5. Write size(). A C++ vector has a function called size(), so our linked list should have the same functionality. We haven't explicitly talked about how to write this, but I'm sure you can figure it out. What is the big-oh time?

6. Write at(). This should work similarly to the C++ vector at() function. What is the big-oh time?

7. Write insert(). This function will allow insertion into the middle of a list.

8. If you have time, look up how to time functions in C++. Write some code to test prepending/deleting the head of a linked list vs the same thing in a C++ vector. Do a million (or more) insertions/deletions and see which one is faster.