**Lab #12**

Linked lists are cool data structures that offer several significant benefits to arrays. Since they grow as they’re needed and don’t need to be allocated ahead of time, they’re really efficient in size and speed of creating and adding items to it. However, since you have to iterate through them from the start to get any data inside, they are slower to search through.

In lecture, you have been learning how to create one from scratch. This lab will introduce you to the Standard Template Library’s (STL) list class, which implements the linked list for you.

**STL Containers**

You have learned about vectors and maps in the past. These are two of the container classes available to you built into C++. More types are deques (doubly-ended queues, or basically a dynamic array), lists (a doubly-ended linked list), and a forward\_list (singly-ended linked list). Just like with vectors and maps, lists have many of the same function names that the previous containers have:

begin() returns the pointer to the beginning of the container

clear() empties a container

empty() returns a Boolean if the container is empty

end() returns the pointer past the end of the container

erase() removes an element at a given position (passed in by an iterator)

pop\_back() removes/pops an element from the back of the container

pop\_front() removes/pops an element from the front of the container

push\_back() inserts/pushes an element into the back of the container

push\_front() inserts/pushes an element into the front of the container

size() returns an integer of the number of elements in the container

and so on. To get the functionality, make sure you #include <list> at the beginning.

Iterators also work the same:

vector<type>::iterator map<type,type>::iterator list<type>::iterator

Strings also are containers of characters. They can be used exactly like the types above, and they also include built-in iterators.

**What You Will Do**

This program should create linked lists from strings, and then remove all of the vowels.

1. Create the file Lab12.cpp.
2. Write a function called **fillList** that takes parameters for a list and the string to add. The function should iterate through the string and add each to the list. (So a string “ABC” should be inserted into a list which will consist of elements [A]->[B]->[C]). **Test this function before continuing.**
3. Write a function called **printList** which takes in a list and an ostream (output stream). It should iterate through a list and print out each element, showing the order they are connected. (From step 2 above, it should print out like “[A] -> [B] -> [C] -> 0”, where 0 is the NULL pointer at the end.) **Test this function before continuing.**
4. Write a function called **removeCharacter** that takes parameters for the list and a character to remove. It should iterate through the list, removing each instance of the character. Think about what happens when you remove characters in a list, and what different types of conditions might occur after doing this. **Test this function before continuing.**
5. In main, read in input.txt, one line at a time. It should stop when it gets to the line “END”. Call fillList to fill the list with the line from the file. After that, make calls to removeCharacter to remove all of the vowels. Then, call printList to print it out to a file called output.txt.

If it helps, draw out some sample lists, and as you trace through your code, think through what the lists look like at any given time. As you step through your program, does your list look like you’re expecting it to?

**Grading and Turning In**

Turn in your program before the end of the lab. Upload only Lab12.cpp unzipped. Grading will be performed using an automated grading program with a different input file (still input.txt) and will be compared to an output file (output.txt). The lab instructor will still manually look at your code for the other grading criteria (functional composition, style, etc.)