# Lab Week 15

## Lists ( STL )

Skills Required

* Standard Template Library
* Create and use classes, Exception Handling, Read and write Files, Work with arrays and vectors, Create Functions, Include Headers and other files, Loops (while, for), Conditional (if, switch), Datatypes, assignment, etc.
* Basic git commands

1. Follow the link that was given on Canvas for this Assignment to create your repository.
2. Use the directions and skills from Week 1 to clone the repository into the directory of your choosing.
3. Open the provided solution in the cloned folder.

**Assignment**

**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**

vector<type>::iterator variable\_name;

map<type1, type2>::iterator variable\_name;

Iterators are ways of traversing through the libraries to go through each element in them. Similar to using a variable to remember a position of an array in a loop, an iterator can remember the position in a library so you can use them easily in loops.

Iterators become pointers to the current element. So as you loop through a vector, an iterator points to the memory location of the first element, then the second, and so on. Since a pointer is a memory address, you can just use the ++ address to increment it to the next element.

For a vector, you can set up a loop like this:

for(vector<type>::iterator iter = vectorname.begin(); iter != vectorname.end(); iter++)

When the loop starts, iter is the iterator name, and it will loop through a vector of type. It starts at vectorname.begin() (which is the first element), and will keep increasing until it gets to vectorname.end(). Each loop it increments itself (iter++).

The iterator must match the type it is iterating through, including the template name. A vector<double>::iterator can’t loop through a vector<int>, for instance.

Since the iterator is a pointer to an element of a vector or map (or really any STL container), to work with the element, you have to dereference it. The way we’ve learned is through the \* operator. Other examples:

If it’s a vector<int>, you can do \*iter = 5;

If it’s a vector<double>, you can do \*iter = 4.85;

If it’s a vector<string>, you can do \*iter = “Vectors rock!”;

If the vector is of a class, remember that

(\*iter).function() is the same as iter->function()

and you can use either one to access the class information.

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. 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.**
2. 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.**
3. 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.**
4. 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?

**Submit your assignment**

1. Update your files on GitHub (Remember to save the files in the IDE before adding them) .sln, .cpp, .h, .vcxproj and any others.
   1. use git status to check which files have been changed and need to be staged
   2. use git add to add the files that have been added or changed
   3. use git commit -m “message” to commit to your local repository
   4. use git push to push this to the remote repository so the grader can find and validate it.
2. Check github repository that your changes and submission are correct. **Show the lab instructor to have them validate the submission.**