COSC 2030

Feb 26, 2018

Practical Introduction to Standard Template Library (STL) Vectors and Iterators

The Standard Template Library (STL) is a collection of C++ code from a compiler vendor that makes everyday programming tasks easier to complete, and greatly improves the quality of the code that we create. In this lab we take a close look at how we can use the `vector` in our own programs, and will take a glimpse into one very useful container sorting routine provided by STL. In addition, we will use a new object called an iterator, and we will see how the idea of a pointer can be "supercharged" to allow for a powerful and useful extension, especially in the context of the STL.

To get started get the following source files: VectorDriver.h, VectorDriver.cpp. These can be found in either the repo shell created for you when you accepted this assignment on Github, OR the Files section on WyoCourses. A couple of new concepts and ideas are implemented in this file, so your TA will take a few minutes to call your attention to the new features, and you should ask lots of questions to ensure that this new material starts to make some sense before you start working on this lab. Begin by separating the program into a header file that contains your functions and a main file that will test your functions.

Lab Assignment

1. The function `fill_vector` assumes that its argument is a vector that already has allocated some space which needs to be filled. We can do this by using the `push_back` method provided by the `vector` class. Look up the `push_back` routine in the .NET help database and ask your TA for help if you want more clarification. Now in your header file implement the `void add_numbers(vector<short> &data)` method that adds 10 random numbers to the `data` vector (feel free to use a `for` loop). Test your code by creating an empty vector in `main()`, call the `add_number` function, and then print the vector from `main()`. Another interesting test you should try is calling `add_number` twice in a row with the same argument - what happens? [L4.P1]
2. Take a look at the sample `print` function provided in the `VectorDriver`, and then create a new function `void print_even(const vector<short>& data)` that prints all of the elements in the `data` vector that are stored at an even index (i.e. data.at(0), data.at(2), ...) Never hesitate to ask the TA for help if you run into problems! [L4.P2]
3. In the tasks above, we used the `at()` method to access elements of a vector. There is another way to do it - through the use of an object called an iterator. For now, think of iterators as very sophisticated pointers that can be dereferenced using the * operator, and can be incremented via ++ (plus, plus) and decremented via -- (minus, minus). What we mean by "incrementing" an iterator is that it moves "forward" in a sequence of items stored in a container. For example, if you start

with an iterator that points to the first element in a vector and then increment the iterator, the iterator will now point to the second element in the vector. There are variations and many exceptions to this basic description (as you might expect), but it should be sufficient to start off our investigation of iterators. Take a look at the `compute_sum` function in the `VectorDriver` for an example of how one might use an iterator. Again, ask questions if things aren't clear. Then implement the `bool is_present(const vector<short> &data, short value)` routine that returns `true` if the `value` is present in the `data` vector, and `false` otherwise. [L4.P3]

4. You might be wondering why use the iterators if we can just use the `at` function. Here is one possible example that may show the usefulness of the iterator objects: the STL provides utilities in addition to data structures - namely useful algorithms that have been implemented for us (hooray!). One of these algorithms is the `sort` algorithm that is able to efficiently sort a container for us in ascending order. All we need to do is tell the sort routine where the start and end of the container are. Guess how we can do that? - Yes, by using iterators. Look up the help files on the `sort` routine. It is defined in the `algorithm` header file, so you want to include this file into your `VectorDriver.cpp`. Now in `main`, manually create a small vector of about 5 numbers in random order, and call the sort routine on the `begin()` and `end()` iterators of your vector. Did it work? What is a very simple test that you can run? [L4.P4]

Turn In:

Upload the **modified and COMMENTED** code files to Github. Make sure your edited version of the VectorDriver.h and VectorDriver.cpp are in the top level of the repo(same folder level as the README).

If a file is not commented so that I can understand what is going on without reference to the instructions I will take off 2 points.
A well commented program will have the programmers name, date, and a description of the file at the top, as well as a block comment before each function describing the purpose of that function. Any other comments explaining individual lines of code are still encouraged.
Grading: 20 points total

	5 pts	0 pts	Total	
L4.P1	add_numbers implemented correctly	Not implemented or incorrect	5 pts	
L4.P2	print_even implemented correctly	Not implemented or incorrect	5 pts	
L4.P3	is_present implemented correctly	Not implemented or incorrect	5 pts	
L4.P4	vector sort implemented correctly	Not implemented or incorrect	5 pts	