CSCI-1200 Data Structures — Spring 2013
Lab 5 — `Vec` Implementation

This lab explores our implementation of the STL `vector` class. Please download:

http://www.cs.rpi.edu/academics/courses/spring13/ds/labs/05_vectors/vec.h
http://www.cs.rpi.edu/academics/courses/spring13/ds/labs/05_vectors/test_vec.cpp

Checkpoint 1

Add a new member function to the `Vec<T>` class named `remove_matching_elements` that takes in a single argument of type `T`, and returns `size_type` (an unsigned integer), indicating the number of elements that matched the argument and were successfully removed from the vector. The order of the other elements should stay the same. For example, if the `Vec<int>` object `v` contains 6 elements: `11 22 33 11 55 22` and you call `v.remove_matching_elements(11)`, that call should return 2, and `v` should now contain: `22 33 55 22`.

If the number of elements in the vector after the call is less than half the allocated size, you should re-allocate a new array that is exactly big enough to hold the current data. (Note that the provided implementation of `resize` member function only makes the vector bigger, so you'll have to implement this functionality).

Add several test cases to `test_vec.cpp` that exercise your new function. Make sure to test various "corner cases", when the element to be removed isn't in the vector, when it's in vector just once, when it's the first element, or the last element, when it's the only element, etc. Make sure it works with integers and at least one other type too.

To complete this checkpoint, show a TA your `remove_matching_elements`, and be present your collection of test cases and discuss why you think this set of test cases is sufficient to debug your function. Also be prepared to discuss the order notation of the `remove_matching_elements` function.

Checkpoint 2

Now, add a `print` member function `Vec` to aid in debugging. (Note, neither `remove_matching_elements` nor `print` are not part of the STL standard for `vector`). You should print the current information stored in the variables `m_alloc`, `m_size`, and `m_data`.

Compile, run, and debug your `remove_matching_elements` function using your set of test cases and add several calls to your new `print` function, to make sure the `Vec` object is reallocated to a smaller array when appropriate.

To verify your code does not contain memory errors or memory leaks, you can use Valgrind and/or Dr. Memory on your local machine (see instructions on the course webpage under "Memory Debugging"). Or submit your code to the homework server (check the button next to lab 5), which is configured to run the memory debuggers for this exercise. To verify that you understand the output from Valgrind and/or Dr. Memory, temporarily add a simple bug into your implementation to cause a memory error or memory leak.

To complete this checkpoint, show a TA your tested & debugged program. Be prepared to discuss the Valgrind and/or Dr. Memory output (with and without a memory error or memory leak).