CSCI-1200 Data Structures — Spring 2018
Lab 5 — `Vec` Implementation

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

http://www.cs.rpi.edu/academics/courses/spring18/csci1200/labs/05_vectors/vec.h
http://www.cs.rpi.edu/academics/courses/spring18/csci1200/labs/05_vectors/test_vec.cpp

Checkpoint 1

Write a templated non-member function named `remove_matching_elements` that takes in two arguments, a vector of type `Vec<T>` and an element of type `T`, and returns 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 v, a `Vec<int>` object contains 6 elements: `11 22 33 11 55 22` and you call `remove_matching_elements(v,11)`, that call should return 2, and v should now contain: `22 33 55 22`.

Add several test cases to `test_vec.cpp` to show that the function works as expected. What is the order notation of your solution in terms of n the size of the vector, and e the number of occurences of the input element in the vector?

To complete this checkpoint, show a TA your debugged solution for `remove_matching_elements` and be prepared to discuss the order notation of the function.

Checkpoint 2

Add a `print` member function to `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`. Use the `print` function to confirm your `remove_matching_elements` function is debugged. Also, write a test case that calls `push_back` many, many times (hint, use a for loop!) and observe how infrequently re-allocation of the `m_data` array is necessary.

To verify your code does not contain memory errors or memory leaks, use Valgrind and/or Dr. Memory on your local machine – see instructions on the course webpage: Memory Debugging. Also, submit your code to the homework server (in the practice space for 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 demo and discuss the Valgrind and/or Dr. Memory output: with and without memory errors and memory leaks *AND* on your local machine and on the homework server.

Checkpoint 3

For the final part of this lab, modify your `remove_matching_elements` function to automatically shrink the size of the allocated array for the Vec object if the number of elements in the vector after the call is less than half the allocated size. Note that the provided implementation of `resize` member function only makes the vector bigger, so you will have to implement this functionality. Fully test this new functionality using `print`, and adding new test cases to test various "corner cases", when the element to be removed is not in the vector, when it is in vector just once, when it is the first element, or the last element, when it is 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`, present your test cases, and discuss why you think this set of test cases is sufficient to debug your function.