# Programming Assignment 1

## Chris Comeaux

## CSCE 221-507

# Program Description

The purpose of this programming assignment was to familiarize students with dynamic arrays and how they are used to construct other data structures, such as vectors. To accomplish this task, the students must write the implementation for a vector class and then create a main function to test all functionalities of the class (My\_vec). After this is done, the students will then create a generic form of My\_vec. This assignment builds the students’ knowledge of how to implement a data structure, how to perform generic programming, and how to create a class.

# Data Structure Description and ADT Run-time of the Class My\_vec

As stated before, the data structure used to create My\_vec is a dynamic array. It is dynamic because when the number of elements equals the total allocated space, the insert\_at\_rank member function will double the allocated space. The My\_vec class has 3 members, 3 non-member functions and many helper functions. Of these functions, many are very efficient and are on the order of O(1) because they all run in constant time. Some of these include: the constructor, destructor, get\_size(), get\_capacity (), the constant access operator, the access operator, elem\_at\_rank(int r) const, is\_empty() const, and replace\_at\_rank(int r, const T& elem). Others are on the order of O(n), such as: the copy constructor, copy assignment, remove\_at\_rank( int r, const T& elem), output operator, and find\_max\_index(const My\_vec<T>& v,int size). This is because they all must traverse the whole vector before they return. Finally, the function that is most complex is sort\_max(My\_vec<T>& vec), which uses a selection sort algorithm. No matter what case is presented to the algorithm, best or worst, it always runs in O(n2) time.

In regards to real time, main.cpp took 331 nanoseconds to run. This was calculated by using the built in clock function in the chrono header file and by comparing the beginning time to the end time.

The My\_vec class uses generic programming so that it can be implemented with any known class that has elements that can be equally compared. It can be used for many things. It can store a large amount of data, find the index of where that max element is, and sort the data. Also, it knows its own size and capacity, can dynamically grow and shrink, interchange its data, and can output its data.

For My\_vec class, a worst case scenario would be if data was inserted at the front of the My\_vec and all existing data would have to be shifted right or if an element was removed at the first index ten all the other elements would have to be shifted left. A best case scenario would be if the element was inserted or removed at the end of the vector and if the capacity would not have to be double. A factor that does affect time complexity is the number of elements in My\_vec. The more elements in My\_vec, the more time it will take to complete some tasks.

# Instruction to Compile and Run Program

To compile the program, use the Makefile provided in the program file. Open PuTTy and navigate to the directory that has the program files. Then type in “make” in the terminal to compile the program. Once it is compiled, type “./main” to run the program.

# Input and Output Specifications

There are no input or output specifications for this program. To test My\_vec yourself, create your own main function, then compile and run it.

# Logical Exceptions

My\_vec will throw a runtime\_error if the main function tries to access, insert, or remove an element that is not in the range of 0 to size-1. Also, if main tries to find the max index on an empty My\_vec, a run-time error will be thrown. One bug in My\_vec is that every time an exception is thrown, the program terminates instead of being able to recover from the exception. The only way to prevent this is by adding a separate try catch block around each function.

# C++ object oriented or generic programming features, C++11 features

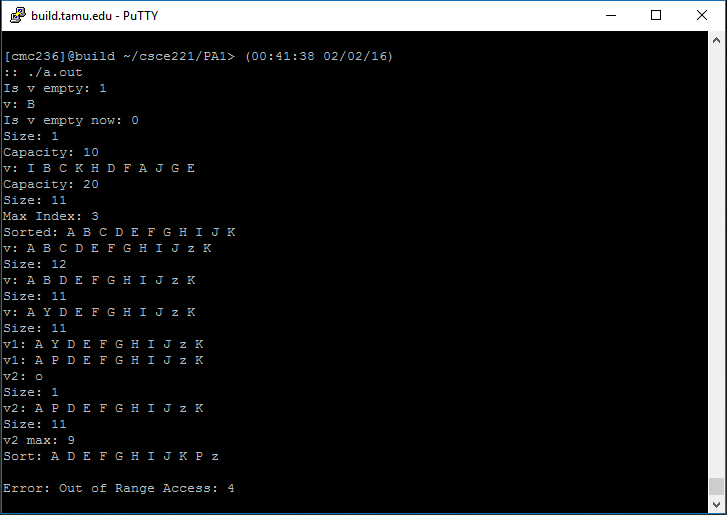
My\_vec is a generic data structure and uses template<typename T>. This means that its elements must be able to be equally comparable. Another generic and C++11 feature is auto. Auto is used in find\_max\_index and sort\_max so that the local variables within the functions can inherit the type of the passed vector.

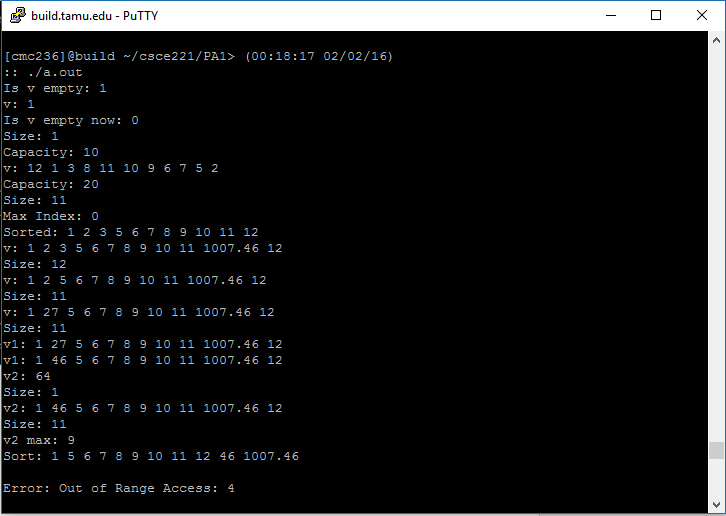
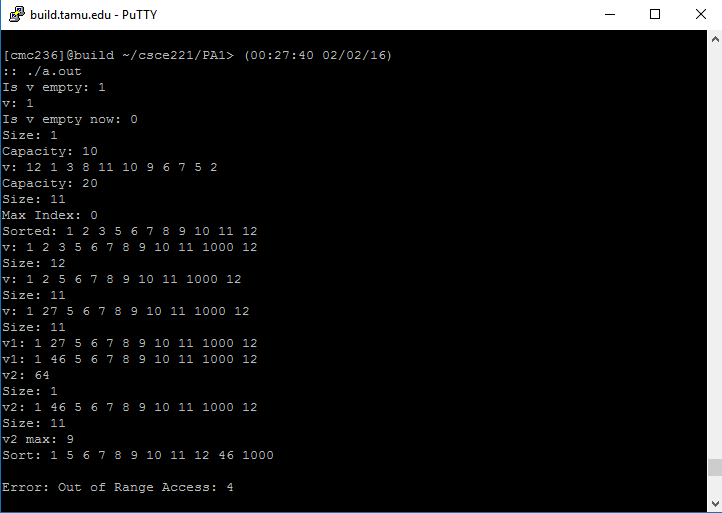
# 

# Testing Results

My\_vec class works as I had expected. However, in the beginning of my testing, I was getting a segmentation fault on my output operator. After some debugging, I found out that it was because I was accessing elements out of range in my access operators. After this problem was fixed, My\_vec worked as expected. Screenshots of the terminal output are listed below:

**My\_vec with chars**





**My\_vec with ints**

**My\_vec with doubles**