Lab 8

Each lab will begin with a recap of last lab and a brief demonstration by the TAs for the core concepts examined in this lab. As such, this document will not serve to tell you everything the TAs will in the demo. It is highly encouraged that you ask questions and take notes. For non-zero labs, you can earn a maximum of 3 points for lab work completed outside of lab time, but you must finish the lab before the next lab. For extenuating circumstance, contact your lab TAs and Instructor.

This week’s lab will be a take-home lab because of Labor day. Please complete the following as homework and be ready to show your TA at the beginning of lab next week to get checked off. You are encouraged to take the lab to TA office hours and ask questions, as you would normally during lab.

Because you will not have as much guidance as you normally do during labs, we have made an effort to anticipate common questions and write their answers into this document. You do not have to go to lab this week, even if your lab time is on Tuesday. The ‘Extended Learning’ section is worth 3 extra credit points. You need to be ready to show the completed lab to your TA at the **BEGINNING of NEXT WEEK’S** lab.

(7 pts) Templated Class and Exceptions

Create your own templated vector class and compare it with the std::vector class.

Copy this **vector.hpp** file and **try_vector.cpp** file. This current implementation doesn’t have a capacity!

vector.hpp:
[http://classes.engr.oregonstate.edu/eecs/fall2018/cs162-001/labs/vector.hpp](http://classes.engr.oregonstate.edu/eecs/fall2018/cs162-001/labs/vector.hpp)

try_vector.cpp:
[http://classes.engr.oregonstate.edu/eecs/fall2018/cs162-001/labs/try_vector.cpp](http://classes.engr.oregonstate.edu/eecs/fall2018/cs162-001/labs/try_vector.cpp)

You need to finish implementing the Big Three, since the vector has dynamic memory.

**Copy Constructor:**

```cpp
vector(vector<T> &other){ ... }
```

**Assignment Operator Overload:**

```cpp
void operator=(vector<T> &other){ ... }
```

Make sure all of this works before moving forward!

**Anticipated question:** What are the member variables for the custom vector class (vector.hpp)?

*v* is a pointer that holds the address to dynamic memory storing an array of type T items. The length of the array is stored as an int in *s*.

(another anticipated question relevant to this section on next page)
Anticipated question: Why isn’t “using namespace std;” used in try_vector.cpp?

If we did use it, the compiler would get confused with the two vector classes (the custom one from vector.hpp, and the one from #include<vector>, which includes “vector” from the standard library).

In order to successfully create instances of both, we can use std::vector and vector now since the namespace std isn’t being automatically used.

(3 pts) Adding Functions and Exceptions

After you have convinced yourself and the TAs that your templated vector class constructors and big three are working, then you can move forward toward implementing the rest of the class. To begin, implement the following functions:

```
T operator[](int);  //Only perform address arithmetic
T at(int);  //Check to make sure not out of bounds
```

In addition, throw an exception from the at() function in the vector template class you created. This function should throw an out_of_range exception, when the user tries to access an element outside the bounds of the vector. You need to add the statement below to at().

```
throw std::out_of_range("out of my vector bounds");
```

First, run your program with trying to access out of bounds memory using the at() function to see what it does now that your function throws an exception, and you are not catching it. Now, catch the exception so that it doesn’t have a run-time error!!! Remember, you can use the what() member function to see your message from the out of range exception.

You will also need to add using std::exception or using std::out_of_range for these types, since we are not bring in the whole std namespace to control which vector we are using!!!

Remember, you and your partner will not receive lab credit if you do not get checked off at the beginning of next week’s lab. You do not have to go to lab this week, you just need to do this assignment at home and be ready to show your TA next week!!

Anticipated Question: How does the push_back function work in the custom vector class (vector.hpp)?

The push_back function takes in a parameter: an element of type T called ele. It then creates a new array of dynamically allocated memory, with a size of ++s (s + 1). “+++s” is like “s++” in that it adds 1 to s, but is different because it adds 1 to s during that line. “s++” adds 1 after the line is executed. (continued on next page)
After creating the array, the function copies all of the elements from the old array (v) into the new array (temp). At this point, all but one spot (the last spot) should be populated in temp.

Then, it deletes the dynamic memory at v (since we already copied this over to temp, it is okay).

Finally, it sets v to the newly created array, temp, and puts the new element to be added (“ele” from the parameter list) as the last element in v.

**Anticipated Question:** How does adding a capacity variable change push_back()?

You will have to check to see if adding one to the size of your array is above the capacity. If it is, you will have to create a new array with a new capacity and copy all of the elements over into the new array (like how push_back() is currently implemented). If not, you just need to add the element to the next array spot (v[s]).

**Because you do not have in-lab guidance this week, the Extended Learning section is worth 3 extra credit points.**

**Extended Learning: How would this change for having capacity and size members?**

What will you have to change in your vector class when you add a capacity private member? The capacity is the actual number of elements allocated on the heap for the vector, and the size is that number that is being used.

**How would the constructors and push_back() function change?** Write out a plan for the extra constructors you might need for testing this and how the push_back() function changes with regard to having both capacity and size members.

Implement these extra constructors and change your push_back() function to operate correctly with capacity and size, now that they may differ.

**Include a resize() and reserve() functionality.**