LAB #7 – Creating C++ Templates, Standard Template Library, & Testing

Remember, if you need to get Lab #4 graded, you need to show your lab to the TAs within 10 minutes of getting to lab, and you and your partner will not receive lab credit if you do not get checked off before leaving each lab. Once you have a zero on a lab, then it cannot be changed because we have no way of know if you were there or not!!!

Reminder: All of our labs involve paired programming. You do not have to keep the same partner for each lab, but you MUST work with someone in each lab!!! First, find a partner for this lab. It can be the same partner from the previous lab or a different partner.

Modify our vector template from class. Copy this **vector.hpp** file (or download) and **test_vector.cpp** file (or download). At this time, you do not need to have a capacity member.

```
template <class T>
class vector {
   private:
      T *v;
      int s;
   public:
      vector(){
         v=NULL;
         s=0;
      }
      ~vector(){
         delete [] v;
      }
      int size(){
         return s;
      }
      void push_back(T ele) {
          T *temp=new T[s];

          for(int i=0; i<s; i++)
             temp[i]=v[i];
          delete [] v;

          s++;
          v=new T[s];

          for(int i=0; i<s-1; i++)
             v[i]=temp[i];
          v[s-1]=ele;

          delete [] temp;
      }
};
```

```
#include "./vector.hpp"
#include <vector>
#include <iostream>

//We do not want to include either stmt. We wouldn't
//be able to compare our vector template to the Standard
//using namespace std;
//using std::vector;
using std::cout;
using std::endl;

int main (){
   vector<int> v;   //Our vector class
   std::vector<int> stdv; //Standard vector

   //Compare operation of our vector to std
   v.push_back(23);
   stdv.push_back(23);

   return 0;
}
```

(3 pts) Big Three

You need to finish **implementing the Big Three**, as well as all the **constructors** you might need, e.g. one that will create a specific sized vector.

Copy Constructor:

```
vector(vector<T> &other){ … }
```

Assignment Operator Overload:

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

Make sure all of this works before moving forward. This requires that you do testing!!! Make a table of data entered and results from execution before moving forward. For example:

Testing What?	Testing Data	Results	Fix
Assign op overload	vector<int> v, other; other = v;	Compile and run successful	
Copy constructor	vector<int> *other = new vector(v);	error: missing template arguments before ‘(’ token	other = new vector<int>(v);

Copy constructor	vector<int> *other = new vector<int>(v);	Compile and run successful	
…	…	…	…

(2 pts) Adding functions to class

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

void resize();

T at(int);  //Check to make sure not out of bounds
```

You probably want to test with types other than ints to make sure the templating part is working correctly!!!! Again, make sure all of this works before moving forward. This requires that you do testing again!!! Make a table of data entered and results from execution before moving forward.

(3 pts) Now, 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.

You probably want to test with types other than ints to make sure the templating part is working correctly!!!! Again, make sure all of this works before moving forward. This requires that you do testing again!!! Make a table of data entered and results from execution before moving forward.

(2 Pts) Now practice using an iterator from the standard vector class.

Create an iterator object to access elements in your vector from the Standard Template Library.

```
std::vector<int>::iterator i; //vector iterator
```

Now, add some elements to the standard vector and print the elements in order and in reverse order using the iterator i, instead of using the standard vector, stdv.

```
i = vector.begin();
```

Once you start the iterator at the beginning of the array, then you can advance the pointer by i++, and it can be an argument for functions to advance it for you, such as insert(). The iterator is a pointer and needs to be dereferenced to access the vector contents.

```
cout << "The contents of a vector element: " << *i << endl;
```

Extended Learning: Add Exceptions to your program…

Modify you program so that it gets a filename from the user to read vector values. Instead of using the **fail()** function to see if the open was successful or not, you can check for successful file opening is by using exceptions. First, you need to turn on the failbit exception, i.e. **input.exceptions(ifstream::failbit);.** Now, try opening a file that doesn't exist. Notice that the program aborts now, if an incorrect file name is provided. In order to handle an exception and exit gracefully is to **try** something and **catch** a possible exception if thrown. Notice, that the exception/error thrown was **failure**. This means that the open will throw a failure exception if it isn't successful. We need to catch this exception, if thrown, and exit gracefully without an abort. **For example:**

```
try {

   //put all the code you want to try in here

}

catch (ifstream::failure e) {

   cout << "The open for the filename provided wasn't
successful!!!" << endl;

}
```

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. The new prototype for the at() function would look similar to the statement below.

```
T at(int) throw (out_of_range);
```