Activity 11

Comp 11 - Summer Session — Getting Organized

With a partner(or two), discuss the following code sample and answer the questions below. The instructor and teaching assistants will let you discuss and then be around to answer questions. [1]

11.1 Description

For this activity, take a look at the following three files. You will notice that it looks familiar! You will paste in your previous lab code into the activity.cpp The interface (activity.h) is already separated for you. Study the syntax.

```
// Include header guards
#ifndef ACTIVITY_H
#define ACTIVITY_H

template <class T>
class CompVector{
private:
    int pos;
    int capacity;
    T* elements;
public:
    CompVector();
    ~CompVector();

    void push_back(T element);
    int size() const;
    T at(int index);
};
#endif // end our header guard
```

Listing 11.1: Interface activity.h

[1] Activities do not need to be returned to instructors, they are for your benefit.

```cpp
#include "activity.h"
#include <iostream>

template <class T>
CompVector<T>::CompVector(){
  pos =0;
  elements = NULL;
}

template <class T>
CompVector<T>::~CompVector(){
  delete[] elements;
}

template <class T>
void CompVector<T>::push_back(T element){
  if(elements == NULL){
    // (1) Allocate memory and update capacity
  }
    else if(pos > capacity-1){
    // Create a new temporary array
    T* new_elements = new T[capacity*2];
    // (2) Copy all of the elements into the temporary
    // array using a loop.

    // Update capacity
    capacity = capacity * 2;
    // (3) Delete our old memory

    // Point elements
    // Why do we need to do this?
    // Answer: ----------------------
    elements = new_elements;
  }

  // Set element to position
  elements[pos] = element;
  // Update position
  pos++;
}

// Return the size of the vector
template <class T>
int CompVector<T>::size() const {
  return pos;
}

template <class T>
T CompVector<T>::at(int index) {
  return elements[index];
}

// Tell the compiler which template to generate.
template class CompVector<int>;
```

Listing 11.2: Implementation activity.cpp

```cpp
#include <iostream>
#include <vector>

#include "activity.h"

int main(){

    // Create our version of vector and the standard libraries
    std::vector<int> cppVector;
    CompVector<int> ourVector;

    // Add some items
    for(int i =0; i < 10; ++i){
        cppVector.push_back(i);
        ourVector.push_back(i);
    }

    std::cout << "cppVector size:\t" << cppVector.size() << "\n";
    std::cout << "ourVector size:\t" << ourVector.size() << "\n";
    std::cout << "cppVector[9]:\t" << cppVector.at(9) << "\n";
    std::cout << "ourVector[9]:\t" << ourVector.at(9) << "\n";

    // Add some more items again!
    for(int i =0; i < 10; ++i){
        cppVector.push_back(i+10);
        ourVector.push_back(i+10);
    }

    std::cout << "cppVector size:\t" << cppVector.size() << "\n";
    std::cout << "ourVector size:\t" << ourVector.size() << "\n";
    std::cout << "cppVector[19]:\t" << cppVector.at(19) << "\n";
    std::cout << "ourVector[19]:\t" << ourVector.at(19) << "\n";

    return 0;
}
```

Listing 11.3: Main Program Control Flow main.cpp

11.2 Questions

1. What is it that we put or define in the interface?
2. Why do we separate the interface from the implementation?
3. What do you think happens if we do not put in the header guards?
4. Ask yourself, will the code be different for a stack than the vector implementation for push? Once you convince yourself of the answer, the important thing to note is that the user of your code will not care, it will just work!