## Homework #2 – Queue Container

In this homework you are asked to implement the following Queue container:

```
template <typename T>
class Queue
{
public:
   Queue();                           // Construct empty queue
   ~Queue();                          // Destructor
   Queue(const Queue &);              // Copy constructor
   Queue &operator=(const Queue &); // Copy assignment operator
   void push(const T &);              // Add elem to back of queue
   void pop();                        // Remove front elem from queue
   T &front();                        // Return ref to front elem in queue
   const T &front() const;            // Return ref to front elem in queue
   bool empty() const;                // Return whether queue is empty
   size_t size() const;               // Return # of elems in queue

private:
   T *v_;                             // Elems in queue
   /* Any other private members you need */
};
```

The public interface of this container matches that of std::queue (except there are no "back" functions). If needed, refer to the std::queue documentation for further explanation of the public interface.

Your implementation of Queue is only required to satisfy these requirements:

- **The queue elements must be stored in the v_ buffer.**
- **All public functions must be exception neutral and strongly exception-safe.**

*The focus of this assignment is to provide a properly functioning Queue implementation that is exception-neutral and exception-safe. You do not need to be concerned with optimizations such as eliminating default constructions of T, minimizing memory usage, pre-growing the element buffer, etc. We will discuss in class whether using an array as the internal representation of the Queue is an ideal solution but for this assignment it is required.*

1. **(3 points)** Implement the following operations (provide at least one unit test for each operation):
   a. `Queue<T>::Queue()`
   b. `Queue<T>::~Queue()`
   c. `Queue<T>::Queue(const Queue &)`
   d. `Queue<T>::operator=(const Queue &)`

2. **(3 points)** Implement the following operations (provide at least one unit test for each operation):
    a. `Queue<T>::push(const T &)`
    b. `Queue<T>::pop()`
3. **(3 points)** Implement the following operations (provide at least one unit test for each operation):
    a. `Queue<T>::front()`
    b. `Queue<T>::front() const`
    c. `Queue<T>::empty() const`
    d. `Queue<T>::size() const`
4. **(1 point)** Make sure your source code is well-commented, consistently formatted, uses no magic numbers/values, follows a consistent style, and is ANSI-compliant.

**Place all source code and a screen capture of the output produced by your program in a single PDF document. Submit this document.**