lab 10: Stacks and Queues

Instructions: Implement a Stack and a Queue with whatever data structure you desire. Remember you may use code from a previous lab (*hint* *hint*).

In class we will implement a stack:

```
1
2  #ifndef STACK_H
3  #define STACK_H
4
5  template<class T>
6  class Stack {
7      private:
8          /* Class to implement.*/
9      public:
10         /* Empty constructor shall create an empty Stack! */
11         Stack();
12         /* Do a deep copy of stack into the this.
13          * Note: This one uses a reference to a Stack!
14          */
15         Stack(const Stack<T> &stack);
16         /* Deconstructor shall free up memory */
17         ~Stack();
18         /* Return the current length (number of items) in the stack */
19         int getLength() const;
20         /* Returns true if the stack is empty. */
21         bool isEmpty() const;
22         /* Print out the Stack */
23         void print() const;
24         /* Pushes the val to the top of the stack. */
25         bool push(const T &val);
26         /* Returns the top element from the stack. */
27         T& top();
28         /* Removes the top element from the stack. */
29         void pop();
30         /* Returns if the two stacks contain the same elements in the
31          * same order.
32          */
33         bool operator==(const Stack<T> &stack) const;
34 };
35
36 #include "stack.cpp"
37
38 #endif
```

In your lab finish the implementation of a Queue:

```
39
40 #ifndef QUEUE_H
41 #define QUEUE_H
42
43 template<class T>
44 class Queue {
45     private:
46         /* Class to implement.*/
47     public:
48         /* Empty constructor shall create an empty Queue! */
49         Queue();
50         /* Do a deep copy of queue into the this.
51          * Note: This one uses a reference to a Queue!
52          */
53         Queue(const Queue<T> &queue);
54         /* Deconstructor shall free up memory */
55         ~Queue();
56         /* Return the current length (number of items) in the queue */
57         int getLength() const;
58         /* Returns true if the queue is empty. */
59         bool isEmpty() const;
60         /* Print out the Queue */
61         void print() const;
62         /* Pushes the val to the end of the queue. */
63         bool push(const T &val);
64         /* returns the first element from the queue. */
65         T& first();
66         /* Removes the first element from the queue. */
67         void pop();
68         /* Returns if the two queues contain the same elements in the
69          * same order.
70          */
71         bool operator==(const Queue<T> &queue) const;
72 };
73
74 #include "queue.cpp"
75
76 #endif
```

Outside Resources:
You may **not** use any outside resources. (IE: No STL, Boost, etc...)

Write some test cases:
Create some test cases, using cxxtestgen, that you believe would cover all aspects of your code.

Memory Management:
Now that are using new, we must ensure that there is a corresponding delete to free the memory.

Ensure there are no memory leaks in your code! Please run Valgrind on your tests to ensure no memory leaks!

How to turn in:
Turn in via GitHub. Ensure the file(s) are in your directory and then:

- $ git add <files>
- $ git commit
- $ git push

Due Date: October 2, 2017 2359

Teamwork: No teamwork, your work must be your own.