CS515 – Lab 4

The purpose of this lab is to introduce you to using Google Test to test a program, and to reinforce your understanding of C++ classes and objects. You will also improve your understanding of C++ memory models.

Download all files from `~cs515/public/4L`

Task 1: Add a test and some assertions to a Google Test source file

Among other things, you are given a file called **`queueTest.cpp`** that has a number of tests, but one test is missing and others are incomplete. You will need to follow the steps below to complete it. You can refer to the file **`queueTest_golden.cpp`** to see how it should look when you are done, but make sure you understand what you're doing—you'll need this knowledge in future assignments.

0. Try to compile (using make) and run `queueTest`. Notice that a couple of tests pass even though there isn't any real code for `queue.h`.
1. Add a test called `constructorCreatesEmptyQueue` as part of the `queueTest` test-suite. Using the `TEST()` macro with two arguments: the test-suite name and the test name. Note that the test name usually says what method is being tested and what should happen.
2. This test is supposed to call the constructor for a queue object and ensure that the object starts off empty. To start, create a local variable called `testQ`. Then on the next line, add an assertion that tests to ensure the size of the new queue is 0:
   ```
   EXPECT_EQ(0, testQ.size());
   ```
   Note that the first value passed to the `EXPECT_EQ` macro is the expected value of the call. Try compiling and running the test to see if it still passes. Try to edit queue.cpp to make the test succeed (it's ok to just return 0 for now).
3. Add an assertion to the same test to make sure a call to `empty` returns true. When checking for Boolean values, you can use the macro `EXPECT_TRUE()` or `EXPECT_FALSE()` with just the statement you want to test against. Try it out again; it should still pass.
4. Many of the tests use the function enqueueValues to enqueue a series of (either even or odd) numbers on whatever queue is passed. The tests `enqueueSixItemsHasRightSize` and `dequeueItemsHasRightValuesAndSize` use this to populate the queue `testQ`, but they don't check anything and therefore the test is useless. Add appropriate assertions to them as indicated in the comments in these tests, and then make sure they fail when run.
5. In the prior step, you were able to test the result of a `dequeue` operation by testing against the variable `result`. In `dequeueAfterMultipleEnqueueDequeue`, you'll test directly against the return value of the `dequeue` operation. There's really not much difference. Go ahead and add the tests specified in the comment, then make sure this test fails.
6. Down in the test `dequeueAllLeavesEmpty`, add an assertion that ensure that an `EmptyQueueException` is thrown when we attempt to dequeuer on an empty queue:
   ```
   EXPECT_THROW(testQ.dequeue(), EmptyQueueException);
   ```
7. Finally, in the test `assignmentWithManyItemsHasCorrectValues`, change the body of the `for`-loop so that it tests that the dequeued value is equal to `i`. When you test it, you should see so many failures, they fill up the screen!

Task 2: Implement the queue ADT using a linked list

You are given the partial header file **`queue.h`**, write the implementation of the queue ADT in source file **`queue.cpp`**. You are allowed to modify the header file, but you may not change the interface to the `queue` class. You should use the completed `queueTest` program to test your implementation.

The queue is an expandable queue, and it is to be implemented using an underlying linked list to store all queue values. The public interface of the queue class is specified in **`queue.h`**. Since the queue uses dynamic memory, the "Big-5" methods *should* also be implemented. However, you will only implement:

- a **default constructor** that creates an empty queue.
- a **copy constructor** to ensure deep copy of objects;
- a **destructor** to avoid memory leaks;
- an **overloaded assignment operator** for deep copy and avoiding memory leaks.

You must implement the queue ADT using a linked list for this lab. You are not allowed to use any STL containers in your implementation.

The expected output for `queueTest` should include the following lines of text among the tests:

```
10 12 14 16 18 20 22 24
9994 9995 9996 9997 9998 9999
testQ values prior to emptying: 6 8 10 12 14 16 18 20 22 24
```

Important: you much check all valgrind memory errors in your programs.

Submission:

Submit the source file **`queue.cpp`** and the header files **`queue.h`** and a **`makefile`**. You may change the provided `makefile` locally, but we will be testing with our own—be sure your `makefile` has something like the following lines to ensure proper compilation with our grading test cases.

```
all: queue.o
queue.o: queue.cpp queue.h
	$(CXX) -c queue.cpp -o queue.o
```

You should also fill out the README file and submit it as well. To submit your files, use the following command on agate (we can't run Google Test on Mimir directly, so the old system will have to do):

```
~cs515/sub515 4L queue.cpp queue.h makefile README
```

- Do not turn in executables or object code.
- Make sure your submission compiles successfully on Agate. Programs that produce compile-time errors or warnings will receive a zero mark (even if it might work perfectly on your home computer.) To check this, use the make command with the provided makefile.

Important:

You must include the standard comment block in each of your source files, including your name, section, date and collaboration details. You must also finish the README file along with your programs.

You should also include detailed collaboration declaration information in your comments. If you worked in pairs in this lab, each of you must include the partner's name in your program comment. Both you and your partner must complete an individual submission in order to earn a grade. If you work by yourself, you must indicate in the program comments that you have worked on your own independently.

You can resubmit as needed—your last submission will be graded. Here is a tentative grading scheme:

directions	100
12 tests in queueTest suite, each worth 5 points	60
Valgrind test: > 300 frees AND < 180,000 bytes lost	10
Valgrind test: > 10,000 frees AND < 20,000 bytes lost	10
Valgrind test: no memory lost (≤ 72,704 reachable)	20
using STL containers	-100