CSCI 262 Data Structures

Spring 2019

Lab 11: Queue, part 2

(Back to Labs)

Goals

- Complete a simple data structure (a queue) using dynamically allocated memory
- Practice implementing the Big 3 on a data structure with dynamically allocated memory
- Practice creating a template class

Overview

The Queue class we created in the previous queue lab had a limited capacity, and could only hold `char` values. This lab will walk you through the steps to make a general purpose, unlimited capacity queue data structure.

This lab will take longer than a normal lab; therefore, you will have extra time to complete the lab.

As usual, we ask that you work with a partner or a team; you don't have to work with the same partner(s) as for the previous queue lab. If you end up solo, please raise your hand and we'll pair you with someone else (or make a three-person team if necessary). You are welcome to change partners throughout the semester!

Before you begin

Decide whether you want to work with your solution for the previous queue lab, or mine. My solution can be downloaded as lab11.zip. You should take a little time to refamiliarize yourself with the basic design and implementation of the queue from the previous queue lab.

For this assignment, you are not provided with tests, as there are several stages to the assignment. You will need to do your own testing for each stage of the assignment, and part of your final grade will be for providing some kind of test code (see below). The test code in `main.cpp` from the previous queue lab can be modified, if you wish, to do some basic testing. Note that the original tests will actually break after phase 1, because the tests assume a fixed size for the queue object. You are welcome to throw out the `main.cpp` altogether and create your own `main()` to test as you go.

In theory, you can do the three phases below in any order, but it is suggested you do them in the order given.

Testing code

You are pretty much on your own as to how you want to test your code for this lab, but you are required to provide us with some kind of `main()` function that we can compile and run to exercise your Queue class. As noted above, you can start with the test code from the previous queue lab, if you wish, but you are not required to use that approach. You may simply write a `main()` that uses your Queue class, testing the various capabilities in your own fashion. The phase descriptions below will detail what kinds of tests you should include for each phase.

Phase 1: Dynamically allocated Queue class

For this phase, you will make it possible for your queue to hold an unlimited number of elements (up to available memory). The approach will be very similar to what we did in building the initial ArrayList class in the ArrayList lecture. This is the hardest of the three phases. **Do the work below in stages, testing as you go!**

To start with, note that we hard-coded the size of our queue in the previous queue lab. You will need to change this and use a private member variable to track your queue *capacity*. (Remember the difference between *size* and *capacity*!)

As soon as you are no longer hard-coding the queue capacity, you must also switch from a static array to a dynamically allocated array. Your array member variable should change to a pointer, and your constructor will need to be modified to set an initial capacity and create an initial array.

Once the queue is working with the above changes, you can implement resizing. Modify your `enqueue()` method to double your queue capacity when you would otherwise run out of room.

Your code for resizing will need to do the following:

1. Allocate a new array using `new`
2. Copy data from the old array into the new array (see note below)
3. Deallocate the old array using `delete[]`
4. Update any other affected variables, like your capacity, front, and back indices

Note that resizing for a circular array based queue is trickier than it was for the ArrayList class in that your data is not stored in a fashion that you can simply copy the existing array into the lower part of the new array, because it is possible your data is wrapped around the end of the array (after a sequence of enqueues and dequeues). The best strategy is probably to move all of your old queue data to the front of your new array, resetting your front index to zero. Test your work on this step carefully!

Once you've completed this step, note that your `enqueue()` method should no longer ever return `false`! (You may change the method to a `void` method if you want.)

Tests:

For this phase, your test code needs to at least enqueue more elements than the initial capacity of your queue, and check that you can dequeue the elements correctly. A more thorough test should be designed to do a sequence of enqueue and dequeue operations that will create a wrap-around in your array, followed by enqueuing to the point of causing a resize.

Phase 2: The "big 3"

Once you've implemented phase 1, your Queue class will no longer behave properly when copied or destroyed. Thus, the next step is implementing the "big 3" for your Queue class, as covered in the Big 3 lecture. Fortunately, these can be implemented very similarly to what was done for our ArrayList class. **Don't just blindly copy code from the slides - make sure you understand what is happening, and apply that understanding to your Queue class implementation.**

Tests:

For this phase, your test code needs to test copy construction and assignment only. (It is very difficult to test for proper destruction without specialized tools.)

Phase 3: Template Queue class

As noted previously, the current Queue class can only hold `char` values. Fix this by turning your class into a class template, as we did for ArrayList in the Templates lecture. Remember that all of your code should be in `Queue.h` after this phase (if it wasn't already)!

Tests:

For this phase, you should replicate all of your test code from the other two phases, but making queues of at least two different types.

Submission Instructions:

Submit a .zip file to Canvas containing your source code: `main.cpp` or whatever you used to test with and `Queue.h` (note that if you completed the "Template queue class" step, you should have no `Queue.cpp` file). Your zip file should also contain your README containing YOUR NAME, the name of your lab partner(s), and any other information you think we should know.

Grading:

Dynamically allocated queue	5 points
The "big 3"	5 points
Template Queue class	5 points
Testing code	4 points
README	1 point
Total:	**20 points**