CISC 3130
Project 0

A Priority Queue

Introduction

Recall from Section 1.2.1 that a *priority queue* uses a queuing discipline which always removes the "smallest element" from the queue. That is, the two operations are `add(x)` and `removeMin()`. In this project, you'll implement and work with a simple priority queue.

Implementation

Using the book's `ArrayQueue` definition, derive two versions of a priority queue. Note that there are essentially two ways (using an array-based structure) to achieve the "remove smallest" operation. The first is to make sure the queue is always sorted, so that the smallest element is always at the front. The second is to leave the queue in its "natural" arrival order and search for the smallest element every time `removeMin()` is called. Implement both of these—call the first subclass `PriorityQueueSort` and the second `PriorityQueueSearch`. Assume that the element type, `T`, has the `<` operator defined. (Be sure your implementations do the sensible thing in the case where two elements have the "smallest" priority—which one should get removed first?)

Timing

To get a sense of the timing, let's just work with `int` values between 1 and 10, as follows:

1. Generate 5000 random values in this range (see below) and store them in an array.
2. Time the process to `add()` them all to `PriorityQueueSort`
3. Time the process to `add()` them all to `PriorityQueueSearch`
4. Time the process to `removeMin()` all 5000 values from `PriorityQueueSort`
5. Time the process to `removeMin()` all 5000 values from `PriorityQueueSearch`

Before you run the code, you should have a sense of the relationships among the timings for 2–5. Are any of those steps going to take more time, or less, than the others?

To generate random numbers between 1 and 10, use the `random` library (be sure to `#include <random>`):

```
// produces random #s between 0 and 1
std::default_random_engine generator;

// converts those to equally-likely ints between 1 and 10
std::uniform_int_distribution<int> distribution(1,10);

int priority = distribution(generator);
```

Toward a Simulation

Of course, we don't put *only* priorities in the queue; we put in objects that represent (for example) hospital patients, or retail shoppers, or computing processes—each of these *have* a priority that we use to decide which gets processed next.

Define a class `Shopper` that includes fields `long int loyalty`, `int cartSize`, and `int priority`. Provide a default constructor that gives `loyalty` and `cartSize` random values, and `priority` 0. Provide an `int` constructor that takes a `priority` value and randomizes `loyalty` and `cartSize`. Provided an overloaded `operator<` for `Shopper` objects that compares based on the value of `priority`. (You might want to overload `operator>` and some of the other relational operators, too.)

Adapt your code from the "Timing" section to work on 5000 `Shopper` objects rather than 5000 `int` values.

Output

Your main program should produce exactly 8 lines of output, like this:

```
priority queue sort add ints: xxx ms
priority queue search add ints: xxx ms
priority queue sort remove ints: xxx ms
priority queue search remove ints: xxx ms
priority queue sort add shoppers: xxx ms
priority queue search add shoppers: xxx ms
priority queue sort remove shoppers: xxx ms
priority queue search remove shoppers: xxx ms
```

What to Turn In

Email me a zipfile named familyname_givenname.zip. (So I would send `dexter_scott.zip`). You may also send a `.tgz` file if you're using Unix/Linux. Do not send a RAR or other kind of compressed file; those will get a grade of 0 points. Make sure the subject of your email is *CISC 3130 Project 0.*The zipfile should contain:

- Your makefile, which produces an executable file called `project0`
- All `.h` and `.cpp` files in your project (including the code from the book).
- A textfile containing 8 lines and 11 columns that records the results of running your program 10 times.. Each of the first 10 columns should contain the output timings from one run of the program; the 11th column should contain the average of the first 10 columns.

All files, excluding the source files from the book, must contain your name at the top.

I should be able to extract files from your zip, run `make`, and run `project0`. If any part of that fails, you will not earn full points.

On Style

You do not need to change the style of the code from the book. But your code should follow best-practice style guidelines—probably Google's C++ Style Guide is the most comprehensive. Pay particular attention to the sections on Naming, Comments, and Formatting. If I have to work hard to figure out what you're trying to do, you will not earn full points.