CS515 – Lab 6

The purpose of this lab is to review Priority Queue ADT and its implementation based on a binary heap.

Note: This lab can be used as the basis for part of assignment 8P—You'll be at a disadvantage if you skip it!

Download all files from `~cs515/public/6L`

Task 1: Implement a **Minimum** Priority Queue based on binary heap.

A Minimum priority queue guarantees that its `dequeue()` method returns the queue element associated with the minimum key value. A minimum priority queue can be conveniently implemented with a minimum binary heap—one that has the *min-heap property*: the value of each node is less than or equal to the value of its children, with the minimum value element at the root of the heap.

The starter code **`PQueue0.h`** declares a minimum priority queue class that stores only `int` elements. We assume the priority queue will hold a maximum of 100 items.

You need to write the implementation in **`PQueue0.cpp`.** The two key methods to write are `moveUp` and `moveDown`.

- The `moveUp` method assumes the heap property is violated at the very last item in the heap array so the item needs to be moved up.
- The `moveDown` method will take an array index position and start "moving down" the item in that position from there. This method is called to move down the root element during the dequeue operation, and also used during the Floyd's O(n) heap construction algorithm.

Here is the algorithm for min heap's **`moveDown(int tmp)`**: while the value at index **`tmp`** is not a leaf and is greater than any of its children, swap this value with its smaller child (if two children nodes have the same key, a random one is selected to be swapped).

Note that in this particular implementation, items are actually stored starting at index 1 in the heap array (for the convenience of index calculation). This is different than examples in class, in which items are stored starting at index 0. This means you will need to use a different set of formulas to calculate the index of child- and parent-nodes of a given index.

Once you are finished, run the given test driver **`testPQ0.cpp`**. You should write your own additional test files to test all the other methods in your implementation.

Task 2: Implement a Minimum Priority Queue class template

Based on your `PQueue0.h` and `PQueue0.cpp` from Task 1, write a class template of the minimum priority queue. The template can be used to create generic priority queues for any comparable types with a given maximum queue size.

The template header file is given as **`PQueue.h`** and the template implementation file to be implemented should be named as **`PQueue.cpp`.** Note in the template declaration that T is a template type parameter and MAX_SIZE is a template non-type parameter that specifies the maximum number of elements in the queue.

```
template<class T, int MAX_SIZE>
class PQueue{
   // . . .
}
```

Once you are finished with the template, compile and the given test driver **`testPQ.cpp`** using

```
%g++ testPQ.cpp –o testPQ      /* compile template driver only */
%./testPQ
```

and you should get the same output as from `testPQ0.cpp` in Task 1. You should also write additional test cases to test your program. You should **not** modify the provided header files while implementing your priority queue.

Submission:

- Submit the files **`PQueue0.cpp`** and **`PQueue.cpp`**
- You may change the 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: PQueue0.o testPQ

PQueue0.o: PQueue0.h PQueue0.cpp
      $(CXX) -c PQueue0.cpp

testPQ: PQueue.h PQueue.cpp testPQ.cpp
      $(CXX) testPQ.cpp -o testPQ
```

You should also fill out the README file and submit it as well. To submit the files in Mimir, follow the link for this lab in MyCourses. Here is a list of files you are expected to submit:

`PQueue0.cpp PQueue.cpp README`

- As always, do not turn in executables or object code, and 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, and check that your submission passes tests on Mimir.

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
PQ test run 1 (non-template)	15
PQ test run 2 (non-template)	15
PQ test run 3 (non-template)	20
PQ test run 4 (template)	15
PQ test run 5 (template)	15
PQ test run 6 (template)	20