COSC 2P95 – Lab Exercise 7 – Section 01 – Templates

Lab Exercises are uploaded via the Sakai page.
Since you need to submit both a sample execution and your source files, package it all up into a .zip to submit.

In Lab Exercise 5, you wrote a simple *Priority Queue*. If needed, refer back to that exercise for an explanation of how a PQ works.

For your Lab 7 task, you took your (regular FIFO) Queue from Lab 5, and made a templated class version.
Can you see where we're going with this?

In this lab exercise, you'll be rewriting your Priority Queue. This time, it must be stored in a specific file, `PriorityQueue.h`, which will contain the entire Priority Queue (i.e. *don't* separate it into a `PriorityQueue.cpp` as well).
This Priority Queue has the following requirements:

- You may implement your Priority Queue *contiguously* (with an array), or *dynamically* (with a linked list), with the following restrictions:
  - If you choose to go with array, don't set the capacity in the constructor. Instead, use an *expression parameter* (refer to Lab 7) to define the length of the array
  - If you go with a linked list, be careful with your memory management. Not only will you require a destructor, but it will likely need a recursive function
- The Priority Queue ***must*** be templated
  - It will require a single typename parameter, to indicate what it holds
  - To that end, though you're free to use structs internally, you won't be using them from the client side, unless what's being held just happens to be structs
  - You'll have to write your `enqueue` method to accept two parameters: a T to store, and a `long` for its priority
  - This should go without saying, but because it's templated to hold *any* type, you obviously can't hardcode any stored types
    - e.g. if you start writing it to hold strings for testing purposes, make sure you remember to delete all occurrences of 'string' from the final version
- As before, lower priority values have a higher priority for being removed
  - e.g. if you add ("eenie", 33), ("meenie", 20), and ("miney", 40), then the first element to be removed will be "meenie"
- In addition to `enqueue` and `dequeue`, also add:
  - A `bool isEmpty()`
  - A `T peek()`, to return what the next *dequeue* will return, but without actually removing it from the Priority Queue
  - An `int count()`, to return the current length of the Priority Queue

There should not be any `cout` (or `cin`) statements anywhere in your Priority Queue class!

Additionally, write a very simple program to demonstrate your Priority Queue.
For this client program, instantiate the Priority Queue to store `std::strings`, and let the user choose to add and remove as many elements as they wish (with whatever priorities they like), until they choose to quit.

Requirements for Submission:

For your submission, in addition to including your source files, also include a sample execution or two.
Pack it all up into a .zip to submit.