Project 4 – Priority Queue using the STL

We find that our priority queues worked well, but now we need to deal with the fact that someone beat us to it.

http://www.cplusplus.com/reference/stl/priority_queue/

Yes, the priority queue is built into queue in the standard template library (STL). So, let's use their interface to get some idea of how to manage priority queues. Alas, there is no provision for handing the ordering of the idea of a Priority String, our favorite object! How will we work around this?

We will do this in two different ways. First, we will create the operations of < and > for **PriorityString**, to see how we can use the support comparators **greater** and **less**. This is unfortunate, as comparing strings normally is by alphabet order, but it will prove that we can do what we expect. It is worth validating in your driver code – you should be able to create a string starting with Z but a low priority, and find that it is less than a string starting with A, but having a high priority. This would not be the expected effects of checking two standard strings with less than and greater than, so we will want to find something better.

Something better, of course, is a proper comparator function. Examples of how one works can be found in the link provided or in the lecture notes for Priority Queue (Week9). Comparison functions work by defining **operator(),** so that is what you will have to do, and figure out how to implement a total-order-relation. Note that implementing one that works in both directions will require a constructor.

Finally, we'll add the non-member functions **operator<<, operator>>,** and **operator+** to priority_queue by simply building a template implementation file. This is possible because we don't need friend status to see the elements. However, this will limit us to the standard methods for queue, so figuring out how to make these functions work in a non-destructive way is critical.

You have been provided some driver code as a basis from which to work, but the rest is up to you. For the driver code the priority order is low to high, but you can do other tests with different rules for order (and you should, to test **greater** and **operator>**). Implementing the Driver code, as you might expect, is required.

Now, you may discover some problems with your comparators using the sample code. This is not unexpected, so you may consider ways to overcome what is happening. If you can, that's great. If you can't, try to explain what you perceive to be the problem, so you can record it as a real effect of using the STL priority queue. Make sure your thoughts are clearly documented.

By implementing the code you will get some experience with the STL, priority_queue, operator overloading, non-member functions, and debugging diagnostics.