partner: **Bin Li** (bzl0024)
partner: **Albert Wallace** (aew0024)

COMP3270 Programming Project
Design & Quick How-to

26 July 2013

Design:

The system consists of two main parts: the heap and the console. The heap is the class that implements the Priority Queue. First, a structure called 'Object' is created to stand for an object in the heap class. It has three fields: Object_ID, priority and index. All of them are zero-based positive integers (must be greater than or equal to 0) except for priority, which is one-based positive interger (it can be set to any number greater than or equal to 1). Each object has only one specified Object_ID, meaning no two objects may share an Object_ID. The higher value of the priority is, the higher priority or weight it has in the heap. In addition, the index field is used for recording the logical position of the object in the heap. According to the project description, each object should be stored in an array. The position in the array is just its Object_ID. That is to say that the Object_ID is the physical position of the object in the array. Thus, in the ELEM structure, an array (which stores all objects physically) is created. However, it is not enough for our project. If we regard the Object_ID as the same index in the heap, a discrepancy arises. For the heap, each object is more likely to change its position in the heap dynamically in order to maintain the max-heap attribute. On the contrary, each object is just fixed in the array with an index of Object_ID. Therefore, each object should have a logical position in the heap called index. When the object needs to change position in the heap, the physical position is not modified any more, but the index would be updated.

The console class is the command interface for users which includes three main commands: enqueue, dequeue and changeWeight. (In addition, the heap is implemented as a private object in the console class. This heap object is used in enqueue, dequeue and changeWeight methods). The command is put into the console as a string via the keyboard, and this string is parsed to determine the desired command. A private array in the console class contains all available commands used for the parsing the input string. After comparing the input string to the elements in that array, if the input is valid, the parser calls the desired method. When the command is executed and the necessary information is returned, the console continues to wait for next command. The system does not stop until an input string of "quit" or "exit". To show the short list of available commands, with a brief explanation of each, use the "help" command.

To run:

Due to weird constructor issues, two separate packages are provided: one that works under Linux (includes consoles and test cases), and one that works under OS X (the console source code is slightly tweaked for the operating system, while the test cases are the same as the Linux versions) (specific versions may make a difference). The OS X package may function under Windows (support isn't guaranteed).

To use the console, compile all associated files (heap.h, console.h, console.cpp; include stdbool.h under OS X) in your chosen compiler. Due to platform differences, compiler differences, and overall fickle compiler issues, no Makefile is provided. (If compiling at the command prompt, using g++ xxx.cpp -o xxx -std=c++0x should suffice to compile each file). Compilation order should be:

- (To use the console under Linux) heap.h -> console.h -> console.cpp
- (To use the console under OS X) stdbool.h -> heap.h -> console.h -> console.cpp
- (To use the test cases under either OS, for each individual test case) heap.h -> testcase____.cpp

After compilation, run as normal. To launch from the command line under Linux, for example, simply run "./[name of executable]" (for example, "./console") with no command line arguments. Keep in mind that the console accepts your commands once it is launched. For other platforms, run as you would any other freshly-compiled file, still with no additional command line arguments.

A number of test cases are provided in different files. Each file (named testcase_____.cpp) tests different functions within the heap class to ensure they are consistent, though they do not use asserts (is it all up to the user to visually verify the test cases). As was the case for the console, each test file should be run without additional command line arguments. There is no user manual for what the output should be, but the desired output is listed as comments in source code (to demonstrate what is expected when the code is called in such a manner).

Additional notes:
Linux variant compiled using g++, on a remote machine. No special flags were necessary.
OS X variant compiled with jGRASP using a "generic" g++ compiler (with default parameters in the settings for jGRASP). OS version was 10.8.4.
Attempts under Windows were done with Windows 7 using Visual Studio Express 2010 (using default parameters).

☺