**CS 2420 – Fall 2014**

**Program 5 –20 points Priority Queues**

**Priority Queue**

We have seen a min priority queue in the past, but you can imagine a max priority queue. A max priority queue is a data structure that allows at least two operations: insert which does the obvious thing and deleteMax, that finds, returns, and removes the maximum element in the priority queue. We also want the heaps to implement a merge operation.

The max priority queue will be implemented three different ways

(1) as a d-heap (a table which logically represents an almost complete d-ary tree) [Code for a binary heap is provided. You just need to convert it to a d-ary heap. Use a value of d=8.]

(2) leftist heap

(3) skew heap.

The test you will use is part of the starter code: TestPQ.cpp. As always, adapt the starter code as you see fit. Note, to facilitate debugging you can control how many things you put in your queues by setting HOWMANY. When you start out, create very small queues that you can trace the execution by hand. You want to KNOW the operations are working properly (not just think it seems to be working). [You are the car mechanic. You don’t want to guess things are working okay. You want to KNOW. ]

Similarly, you can limit how much of your Priority Queues print by setting the variable PRINTSIZE in the file. For the D-heap, you will just print the first specified (PRINTSIZE) number of elements in the array. For the leftist heap and skew heap, PRINTSIZE will limit the number of levels to print. PRINTSIZE will indicate a specified number of levels of the tree (where the tree is printed prettily. indented to show nesting level).

The input file consists of (word,frequency) pairs which indicate the frequency of word use in some set of documents. We are interested in the most commonly used words. Note that keys with the same priority value may be deleted in different orders, depending on implementation differences.

Each data structure needs to be able to convert its contents into a string using a toString() method you create. Code for the leftist heap and the skew heap is to be original (not copied from ANY other source). All the heaps should have the same interface defined in PQ.h. You can change the interface if you like.

Compare the results of the three methods of implementing a priority queue, and submit your comparison in a file named “readme.txt”. For each method, answer the following questions. [Count operations loosely – but try to compare similar work between the methods. ]

a. On average: how many compare operations were required to do an insert? How does your count compare to log n?

b. On average: how many compare operations were required to do a merge? How does your count compare to log n?

c. On average: how many compare operations were required to do a delete min? How does your count compare to log n?

When you have completed the assignment, zip your entire project and submit the zip file to Canvas. Summarize your implementation results in a “readme” file which is submitted with your project.

**Hints**

Notice that if all the queues extend PQ, we can use a different type of implementation without affecting Test.cpp.