CSC 270: Assignment 3

Due: at beginning of tutorial, Fri. Nov. 14, 1997.

1. **Heaps and C++.** I have provided most of the code necessary for a simple implementation of a heap in C++. The source code and a Makefile is available from `tuzo.erin` in the directory `~jepson/270/srcpp/heap/`.

a) Read the source file `brokenHeap.cc`, taking note of the two places you need to insert code. Make and run the code before you alter it. Describe, in your own words, the algorithms push and pop actually implemented by `brokenHeap.cc`. (Note: These algorithms do NOT provide a correct implementation of a heap.)

b) Fix the implementation of push and pop for heaps by inserting code after the large C-style comment blocks (i.e. starting with /* and ending with */) in the source file `brokenHeap.cc`. (About 40 lines of code, in two blocks, were removed from a working implementation to make `brokenHeap.cc`, so you shouldn't need to add much more. Also, none of the existing code needs to be changed.) Run your modified code on the example provided in the main routine of `brokenHeap.cc`.

c) Change the main routine and `echoHeap` so that they generate a heap of `events` instead of `testItems`. Each event should have the fields `time`, and `eventName`. Show how the heap could be used to push events onto it, and pop off the event(s) which occur at the earliest time. Make up your own test case.

d) Describe, but do not implement, an algorithm for removing an arbitrary element of the heap (not just the root). Assume that you are given the index, `i`, of the element, `heap[i]`, that is to be removed from the heap. After the removal of `heap[i]`, the tree associated with the heap should still be partially ordered (i.e. parents have lower priority than their children) and left justified (i.e. all empty spaces in the heap occur after the last item, namely `heap[last-1]`, in the array). How many comparisons does your algorithm require?