Notes on cs2223 Project 3
Implementing a Securities Market[*]

Due Tuesday, 20 Nov 2012

Here are some notes on the goals and implementation strategy for project 3 on implementing a security market via priority queues.

Important. Be sure you see the note below on *Test cases and results.*

Modularizing the priority queues. One of the main jobs in this project is modularizing the priority queues. You will adapt the code from `min_heaps.lua` so that it can be used with any comparison function.

The comparison function can in fact be any function $f(a, b)$ of two arguments that returns true or false. The function $f(a, b)$, when it returns true, says that b must not appear above a in the heap.

The code that you end up with should work properly as long as the function f is transitive and antireflexive. "Transitive" means that as long as $f(a, b)$ returns true and $f(b, c)$ returns true, then $f(a, c)$ will also return true. "Antireflexive" means that $f(a, a)$ does not return true. This is important because a priority queue could have two entries with the same value, and so we will allow either to occur above the other. The criterion for "working properly" is that the code should ensure that the heap representation always satisfies its *invariant*, namely:

- If p is the entry in a parent node and c is the entry in one of its child nodes in a heap, then not $f(c, p)$.

When $f(a, b)$ is the condition $a < b$, this gives us a min-heap, and when its the condition $a > b$ it gives a max-heap.

In this project, you will build heaps that store two different kinds of objects. These are the buy offers and sell offers for a security. Notice, neither of these is a number. They're both *tables* with several fields. So a comparison function like $<$ or $>$ wouldn't work; you wouldn't want to compare tables numerically.

But you can compare buy orders according to their *bid* fields, and sell orders according to their *quote* fields. So to make this work, you'll need to have some priority queues that act as max-heaps relative to the bid fields of buy orders, and some that act as min-heaps relative to the quote fields of sell orders.

[*]Joshua Guttman, FL 137, `mailto:guttman@wpi.edu`. Include [cs2223] in the Subject: header of email messages. Due midnight at the end of 20 Nov.

How can you do it? First, you'll change the definition of `make_empty_heap` so that it takes an argument that will be the comparison function for the heap it's building. And insert that function into a field of the heap, just like `heap_bound`. Second, change the comparisons in the file—the ones that are comparing heap entries to see which belongs closer to the root—to use this field of the heap. Third, write your client code, the code that's implementing the market, to build heaps with the right comparison function.

In terms of Lua syntax, you can write things like:

```
make_empty_heap(function (table1, table2)
                    return table1.bid>table2.bid end)
```

where the syntax `function (`*args*`) return` *v* `end` actually creates a function value to deliver to the new heap.

Modules and "require" in Lua. Since there were some reports of the old style of modules not working in the distributions some people had installed, I have switched to a new style. You'll have to change lines saying:

```
require "file"
```

to say (choosing some `name` that's convenient for you)

```
name = require "file"
```

Then refer to a function `fn` defined in `file.lua` in the form `name.fn`.

Test cases and results. To make the grading fast and reliable, you must include code at the bottom of your file to run your code on test cases. You must include a block comment with your results on those test cases included.

The results must be understandable from the text you print out. You will need to write a function to print out the transaction log in a readable form, so that the graders can see who bought what from whom at what price.

The built-in `print()` function prints `table: 0x100132cb0` or the equivalent for a table. That does *not* meet this requirement. You need to write your own routine to print these items, showing the fields that matter.