Priority Queues and Refactoring

10-12-2011
Do you think you should be allowed a 1-page cheat sheet for the exam?
Priority Queue ADT

- A priority queue has the same methods as a normal queue, only the contents are ordered not only by arrival time, but also by a priority. So dequeue gets the highest priority object and if several have that priority, it gets the one that has been there the longest.

- One way to implement a priority queue is with a sorted linked list. To make this flexible, you could have it take a comparison function that tells you the ordering. That would be provided when the priority queue is constructed. Or require Ordered.

- What order are the various operations for this implementation of a priority queue?
Let's write a priority queue that uses a doubly-linked list with a sentinel.

We'll also make a trait because we will implement other versions later.
Refactoring

- This is something that you do when you don't want to change the functionality of your code, but you want to change how it does something.

- You typically refactor your code when it “smells.” Here are a few of the many different smells.
  - Long method
  - Large class
  - Duplicate code
  - Shotgun surgery
  - Switch statements

- Scala tools don't yet refactor well, but the language does.
You should have learned about recursive functions in 1320. A recursive function is simply a function that calls itself.

You can use recursion to imitate loops, but we won't do that very often in C/Java/Scala. Where recursion comes in really handy is when a function needs to test more than one alternative at a time.

This works nicely because the call stack remembers where you are in a given function so when you return back, you can take off from that point again.
One of my favorite recursive algorithms is maze solving. This is a special case of graph traversals which are common problems in CS.

We'll use a 2D array of Ints as our maze and we can even put this into our drawing program.

I want to write code to find the shortest path through a maze or count all paths through a maze.

We can try to make this nice and graphical as well so it fits properly into our drawing program.
Another one of my favorite recursive algorithms is formula parsing. This allows us to have the user type in a function and our code can evaluate it. We do this through “divide and conquer”. We split the formula in two across the lowest precedence operator then recursively evaluate the two halves. We can use this to put function plotting into our program if we give it the ability to handle a variable.
Can you think of uses for priority queues in your project?

Review session on Sunday. I'll send an e-mail with the time.