CSCI 5150 **Fall 2014**
Homework 2 **Due: October 3**

In this assignment, you will make changes to Homework 1 to accommodate the new requirements below. You will be creating a peer-to-peer system, where every node is both a client and a server. Make sure that you fix any problems noted in Homework 1.

Combine the two programs from Homework 1 into one program. When the main program starts up, it should start the client code in its own thread; the server part of the program is similar to Homework 1. Make sure that the server part starts up correctly *before* starting the client code, though: if the server fails, just exit and display an error message. Make sure that you label any screen messages as either "Server: " or "Client: ".

The server will not use the command line parameter for the port number. Instead, it will use a setup text file called `portnums` that will contain a two port numbers in it – read these two numbers in as the lower bound (that is, this will substitute for the value in the `-p portnum` argument in Homework 1) and upper bound (this is new – this should be the upper value for the loop) for your portstepper code from Homework 1, and use the first available number. As before, display the port number actually used.

For the client part, the command line parameters are no longer used, either. Instead, use one of the nodes in the list of connections (implemented in Homework 1). This list should now be stored in a text file called `neighbors` on exit. If the file does not exist or it's empty, prompt the user for the name of a server and port number. When a client sends the first message (type 0) to another node, make sure that you send its name in the domainName field (use `gethostname(char* name, int nameLength)`) and its server's portnumber (from the portstepper) to the other node; that other node should record it in its list. Be sure to only record these requests once in the list – a node might get several connections over time from the same client, so check the list to make sure there are no duplicates.

The client part should present a simple menu of choices on standard input, asking for either a query or to exit. Simply send a sample query to the other node and get a sample response. When the client quits, the server does not – it still needs to be killed manually.

You can work in pairs for this program. Email me your zipped code at barrettm@etsu.edu.