LAB 12/2/2014
HW3 starter code

Please begin by downloading the files and opening a graph.cpp file. Recall that the graph.h file defines graphs as a vector of vertexes, and a map of vertexes to their neighbors:

```
        private:
                // list of cities
                vector<vertex> vertices;
                // neighbors of each city (pointers back into the vertices array)
                map<vertex *, vector<vertex *> > edges;
```

The first function in graph.cpp you need is the constructor. It's very easy; the empty vector and empty map get built automatically. (This function is now done.)

```
graph::graph()
{
}
```

The next function is the destructor. Use the clear() function to free up the memory for the vector and the map. 2 lines of code.

```
graph::~graph()
{
}
```

Next, write a function to place a new vertex on the vertices vector, adding to the back. One line.

```
void graph::add_vertex(const vertex& v)
{
}
```

And write a function to place an edge from vertex u to vertex v into the edges map, if the distance from u to v is within the limit. Use great_circle_distance() to get the distance. Watch the types for the edge map, and remember how to get the address of a variable here.

```
void graph::add_edge(vertex* v, vertex* u, double limit)
{
}
```

The last and hardest part of the job for your lab is to read in all 119 cities from the world_cities.txt file. A city's data looks like this:

```
        La Paz, Bolivia: 16 27 S 68 22 W
```

This is the function that reads them in:

```
void init_graph_from_file(graph& g, const string& filename, double limit)
{
        string city_name;
        int lat1, lat2, long1, long2;
        char compass_dir;
        ifstream file_to_read;
        // open the file
        open_for_read(file_to_read, filename);

        // while there is still stuff to read in the file
        //        read in each line of the file, using getline
        //        (http://www.cplusplus.com/reference/string/string/getline/)
        //        get the city name, lat degrees, lat minutes, long degrees, long seconds
        //        the string find and substr functions will be helpful
        //        create a vertex from these using the vertex constructor
        //        add this vertex to the vertices vector

        // close the file when done
        file_to_read.close();

        // Compute the distance between u and v and add an edge from v to u
        // (and from u to v) if the distance is less than limit
}
```

The code I gave you above opens the file. You need to figure out the code so that it can read in what it assumes is a city name till it sees a colon (this marks the boundary between the name and the latitude/longitude data). Then it reads in the latitude degrees, the latitude minutes, and the compass direction (N, S). If the direction is N, then the latitude numbers are flipped to negative numbers. It then reads in the longitude degrees, the longitude minutes, and the compass direction (W, E). If the direction is E, it also flips the longitude numbers to negative ones. This lets us tell the difference between places that are, say, 10 degrees north of the equator and places that are 10 degrees south of it.

Finally, when you have all that information in hand, you call our little vertex constructor, make a vertex, and add it to the graph. Here is that code from the init_graph_from_file() function in graph.cpp:

```
        vertex city(city_name, lat1, lat2, long1, long2);  // constructor call
        g.add_vertex(city); // this code just puts this vertex at the end of our vertices array
```

Now your graph has one more city in it. Read this info in for all the cities. The code above closes your file before it ends.

Finally, we compute the distances between every city and every other city (using a nested loop). You have the code to compute this distance for any 2 cities. When you have a distance below the limit (but not equal to 0), add an edge between the 2 vertices, using the add_edge function.

After this works, you are ok for this week's lab; plan to carry out the depth and breadth first searches to finish hw3.