CSCI 2270 – Data Structures and Algorithms

Instructor: Godley

Due Friday, July 27th by 11:59pm

A Nation Divided … by Zombies!

Admit it, we all love zombies. Maybe it's because they don't actually exist, and we don't actually have to worry about navigating a life among the undead. But, imagine for a second an alternate universe where they do exist and they have attacked – creating mayhem throughout the country, knocking down communications towers and taking control of bridges and highways. One could imagine a resourceful zombie coalition making it impossible to travel between major cities, isolating human survivors in small districts around the country with no safe means of reaching other districts. The US would become a collection of small outposts, where cities within a district could be reached from within the district, and district residents would need to be careful about travel even within their district. Knowing the shortest path between cities to avoid being attacked would be paramount for survival.

What your program needs to do:

Build a graph. There is a file on Moodle called *zombieCities.txt* that contains the names of 10 cities and the distances between them stored as an adjacency matrix. Cities that still have roads connecting them that aren't controlled by zombies have a positive distance in the file. Cities that have been cut off from each other have a -1 as their distance. When the user starts the program, read in the cities and distances from the text file and build a graph where each city is a vertex, and the adjacent cities are stored in an adjacency list for each vertex. Make sure you read in the *zombieCities.txt* file in order so that your vertices are set up in alphabetical order.

For this assignment, you are building the graph, and implementing functionality to determine if two cities are adjacent. In the next assignment, you will add additional functionality to find the shortest paths through the graph and determine if a certain path even exists.

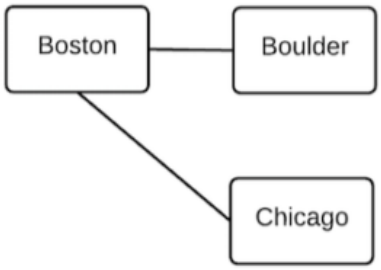
Use a command-line argument to handle the filename.

Your program needs to open the *zombieCities.txt* file using command-line arguments. When we test your code, we will use a file with a different name.

In the adjacency matrix in the text file, you might see data that looks like this:

	Boston	Boulder	Chicago
Boston	0	2000	982
Boulder	2000	0	-1
Chicago	982	-1	0

That matrix would generate this graph:



The vertices in the graph are Boston, Boulder, and Chicago. The adjacent vertices for Boston are Boulder and Chicago. The adjacent vertex for Boulder is Boston, and the adjacent vertex for Chicago is Boston.

Display a menu. Once the graph is built, your program should display a menu with the following options:

1. **Print Vertices**
2. **Vertex adjacent**
3. **Quit**

Menu Items and their functionality:

1. **Print vertices**. If the user selects this option, the vertices and adjacent vertices should be displayed.

   An example of how the output should be formatted is shown here:
   Boston->Boulder***Chicago
   Boulder->Boston
   Chicago->Boston

2. **Vertex adjacent**. This option takes two vertices as user inputs and displays True is the vertices are adjacent, and False if they are not adjacent.

Additional information

There is sample code on moodle showing how to use vectors and add vertices and edges to a graph. Feel free to use that code as the starting point for this assignment. Do not modify the *Graph.h* file. For more information on Vectors, there is a great tutorial here:

http://www.codeguru.com/cpp/cpp/cpp_mfc/stl/article.php/c4027/C-Tutorial-A-Beginners-Guide-to-stdvector-Part-1.htm

Appendix A – cout statements

1. Print menu

```
cout << "======Main Menu======" << endl;
cout << "1. Print vertices" << endl;
cout << "2. Vertex adjacent" << endl;
cout << "3. Quit" << endl;
```

2. Print vertices

```
cout<< vertices[i].district
<<":"<<vertices[i].name<<"-->";
for each adjacent vertex:
        cout<<vertices[i].adj[j].v->name;
        if (j != vertices[i].adj.size()-1)
                cout<<"***";
```

3. Vertex adjacent

```
cout << "Enter first city:" << endl;
cout << "Enter second city:" << endl;

Cities are adjacent:
cout << "True" << endl;

Cities are not adjacent:
cout << "False" << endl;
```

4. Quit

```
cout << "Goodbye!" << endl;
```