Program 3 - Driving Directions

Due Date: 11:59 p.m., May 11, 2016
All public interfaces are minimum requirements. You may add additional public methods if needed as long as you follow the lab and style guidelines.

For our final project you are going to create a library that reads in a file with a list a cities and the roads between the cities. The driver code will input a starting city, and destination city, and you will return the shortest path between the two as a vector of cities along the path.

Part A: Setting up

For Part A you will read in a text containing cities and their x and y coordinates along a grid. The file format will be as follows:

```
paris 6 7
london 3 5
rome 4 7
dublin 3 2
barcelona 4 2
```

City names will not contain whitespace and will be uniquely named. Input files will be well-formed (no need for error checking). You will need to build an undirected, weighted graph using the x and y coordinates.
The user can only travel in straight lines, so paths are along the x and y grid. In other words, there is a path between two cities if they have equal x coordinates or equal y coordinates and there is no city between them on the path. A city will have, at most, 4 adjacent cities. The adjacent city in any one direction is **only** the closest city in that direction. So the algorithm for building the adjacency list will be to find all cities in one direction, but set only the closest one as adjacent.

For example, if you have 3 cities along the same x axis A[5,3], B[5,6], and C[5,4], A is only adjacent C; however, C is adjacent to B in the north direction, and A in the south direction.

You should use an array or vector to store the cities. To hold edges you will need to implement an adjacency list (not an adjacency matrix) using a linked list (you should use the STL list class).

- City
  - Public Methods
    - City(string cityName, int xCoor, int yCoor);
    - std::string getName();
      - Returns the city name
    - int getXCoor();

                - Returns the x coordinate
            - int getYCoor();
                - Returns the y coordinate
            - list<City*> getAdjacent();
                - Returns a copy of the adjacency list
            - bool operator<(City &c);
                - Compares cities by name using the string operator< method

- Map
    - Public Methods
        - Map(string filename)
        - City* findByName(string cityName);

/************Must be completed by end of lab next week (May 5)**************/**

Part B: Finding the Shortest Path

You will need to implement Dikjstra's shortest path algorithm to determine the shortest path between two cities. You must travel in straight lines, and can only change directions at cities. In other words, if cityA has coordinates [5,7] and cityB has coordinates [6,2], there is no path between them. However, if you add cityC with coordinates [5,2], you now have a path between cityA and cityB through cityC.

You should add the following public method to your Map class:

- vector<City *> shortestPath(City * start, City * dest)
    - The method should return the shortest path between two cities by returning a list of the cities you will need to travel through.

You will also need a second method that gives the distance between two points on the graph:

- unsigned int pathDistance(City * start, City * dest)
    - The method should return the total distance (based on the path you must take) between the two cities.
    - The method should return -1 if there is no path.

Part C: Testing your Code

Lastly, you should create your own test file with cities and coordinates to test your code. The requirements for your test are as follows:

- You should have at least 10 cities with coordinates in your files.
- The grid must be at least 25 x25, although I would suggest making it larger.
- All city names must be unique

        - Add your own code to the driver code to run and test your file

Add 3 tests to the driver code to verify the correct path was taken.

Part D : Code Organization and Submission

- Required code organization:
    - program3.cpp
    - Map.cpp/.h
    - City.cpp/.h
    - makefile
        - **Your makefile must include c++11 extensions**
    - readme
- Create a readme file following the format provided, and add it to your project folder
- While inside your program3 folder, create a zip archive with the following command
    - zip -r program3 *
        - This creates an archive of all file and folders in the current directory called program3.zip
        - **Do not zip the folder itself, only the files required for the lab**
- Upload the archive to Mimir under 'Program 3'

Tests

- **Test Driver**
    - program3.cpp
- **Test cities files**
    - townlist.txt
    - Your test file.

Grading Guidelines

- **Part A**
    - Test 1: 1 point
    - Test 2: 3 points
- **Part B**
    - TBD
- **Part D:**
    - Follows formatting guidelines, requested project structure and naming conventions, and submission does not include .o files or binary (1 point)

Formatting Guidelines

- Stores all values in a named variable.
    - No Magic Numbers.
- Uses indentation to identify code blocks.
    - Every Code block should be indented from it’s parent block to identify scope.
- No single letter or non-descriptive variable names
    - The only exception to this rule is ‘i’ in a for loop
- Separates code blocks and logical sections with whitespace
    - Optimize your code for the reader, not the writer
- Output is formatted with an explanation of the output values
    - Format your output so that someone who does not know what the program is supposed to do would know what the output meant
- Each method is preceded by a comment explaining what the method does
- Each significant code block is preceded by a comment explaining what the code block does.
    - A significant code block is more than 3 lines performing a single logical operation
- CONSTANTS are in all caps
- Only data types start with a capital letter
    - Classes, Enums, Structs, etc.
- Do not use the ‘using namespace’ declaration in a header (.h) file
- In general we will follow the Google C++ style guidelines. If you want more info, you can view them here: https://google.github.io/styleguide/cppguide.html