CSCE 2110
Fall 2015
Program 3 – Roads, Distances, & Min Spanning Tree
100 Points
Date Due: Wednesday, November 4, 2015 (Start of Class)

Program 3 for will serve as an incremental implementation of a Road Atlas and Driving Distance Calculator. Specifically, Program 3 adds labels and weights to the basic graph provided in Program 2. In addition, it adds the capability to make the graph DIRECTIONAL by REMOVING specific edges (one way). Finally, Program 3 implements Krushkal's (Minimum Spanning Tree) algorithm.

User input will be in the form of commands. The user must be able to input commands via the command line OR from a file in a CSV format. **BOTH input methods must be available to the user.** This format will allow the easy creation of a large number of inputs quickly without the need for a long series of prompts and responses. The commands used in the CSV file are identical to the command line commands EXCEPT, of course, each field of the command in a CSV file is separated by a comma. Also, neither the command "file" nor the command "Quit" will be issued from a CSV file. These are NOT "error conditions" that must be handled ... it is just a promise from the instructor. Once all commands in a given command file have been processed, the program returns to the interactive mode with a command prompt.

Program 3 must build an Undirected or Directed Graph from user input. Nodes are created first. For the sake of simplicity, all nodes are assumed to be "cities" with the names NA through NZ (26 possible cities). Edges are then created between existing nodes, along with an EDGE NAME and a DISTANCE between the nodes. In accordance with the instructions of Program 2, adding an edge implies that the edge is initially 2-way. Program 3 then provides a command to DELETE specific instances of the edges, possibly making them ONE WAY. Deleting both directions (with two DELETE commands) DELETES an edge, completely. Your program can handle the graph either as an ADJACENCY MATRIX or as an ADJACENCY LIST. The only restriction is that your entire Team must use the same method, since the final program in the class will be submitted as a group project (one submission per team).

Program 3 also adds the command "Krushkal" to produce the minimal spanning tree. Since Krushkal's Algorithm only works on UNDIRECTED graphs, you have to implement a GLOBAL VARIABLE called "UNDIRECTED" which will initially be set to TRUE. As long as this variable is TRUE, Krushkal's algorithm should work. When the user deletes even ONE edge (making the graph DIRECTIONAL), the UNDIRECTED must be set to FALSE. The first thing your Krushkal's function should do is check the value of UNDIRECTED. If it is FALSE, simply print an ERROR message to the user stating that KRUSHKAL's ALGORITM IS NOT AVAILABLE FOR DIRECTED GRAPHS.

When the program is executed, it begins with a command prompt: **cmd>**. Your Program must implement the following commands:

NOTE: Notes from Program 1 still apply.

1. **<u>Add a node</u> to the graph.** The user provides a NAME for a node. If the NAME does not exist in the graph, the node is added and acknowledged. If the NAME already exists in the graph, an error message is issued.
    - FORMAT: **node add <name>** ==>
        - EXAMPLE: **node add NB**
            - Return: **ADDED: NODE NB**
            - Error: If the "name" is a duplicate of one already in the graph, print the following error message to the screen: ***** ERROR *** DUPLICATE NODE: NB**

2. **<u>Delete a node</u> from the graph.**
   The user provides a NAME for a node that should be deleted.
    - **FORMAT: node delete <name>**
        - EXAMPLE: **node delete NB**
            - Return: **DELETED: NODE NB**
            - Error: If the "name" is NOT in the graph, print the following error message to the screen: ***** ERROR*** NODE NOT FOUND FOR DELETION: NB**
        - The deletion of a node may cause the deletion of 1 or more edges. Report EACH EDGE that is automatically deleted as a result of a node deletion.

- **EDGE AUTO REMOVED BY NODE DELETION: NB-NA**
- **EDGE AUTO-REMOVED BY NODE DELETION: NB-NB**

3. **<u>Search for a node</u> in the graph.** The user provides a NAME of a node for which to search.
   - **FORMAT: node search <name>**
     - EXAMPLE: **node search NB**
       - Return: **NODE NB: FOUND**
         NODE NB: NOT FOUND

****
4. **<u>Add an edge</u> to the graph.** ******** DIFFERENT THAN PROGRAM 2 ********
****

The user provides TWO NODE NAMEs as the end points of a new edge, along with an EDGE NAME and an associated DISTANCE between nodes. If the NAMEs exist in the graph without an edge between them, the edge is added and acknowledged. If the EDGE already exists in the graph, assume that the edge NAME and DISTANCE are being modified with new data (even if identical). IF EITHER NODE does not exist, an error message is issued. **The graph is undirected, so adding NB-NA also must add NA-NB.**

- FORMAT: **edge add NODE1 NODE2 NAME DISTANCE**
  - EXAMPLE: **edge add NA-NB S99 100**
    - Return: **ADDED: EDGE NA-NB NAME: S99 DIST: 100**
      ADDED: EDGE NB-NA NAME: S99 DIST: 100
    - Error: If the EDGE is a duplicate of one already in the graph, print the following: **EDGE NA-NB and NB-NA UPDATE: NAME: S99 DIST: 100**
    - Error: If EITHER node does not exist in the graph, print the following error message to the screen: ***** ERROR *** NODE NA: NOT FOUND**
      -OR- * ERROR *** NODE NB: NOT FOUND**

****
5. **<u>Delete an edge</u> from the graph.** ********* DIFFERENT THAN PROGRAM 2 *********
****

The user provides TWO NODE NAMEs as the end points of an edge to be deleted. If the NAMEs exist in the graph with an edge between them, the edge **IN THE DIRECTION GIVEN** is deleted and acknowledged. If the requested edge does NOT exist in the graph, an error message is issued. The global flag **UNDIRECTED** must be set to FALSE to inhibit future execution of the Krushkal's function.

- FORMAT: **edge delete NODE1 NODE2**
  - EXAMPLE: **edge delete NA NB**
    - Return: **DELETED: EDGE NA-NB**
    - Error: If the EDGE does not exist in the graph, print the following error message to the screen: ***** ERROR *** EDGE DOES NOT EXIST FOR DELETION: NA-NB**

****
6. **<u>Find the Minimal Spanning Tree</u> for the graph.** ********* NEW COMMAND *********
****

Execute **KRUSHKAL's** Algorithm to create a Minimal Spanning Tree for the current graph IF and ONLY IF the global variable **UNDIRECTED** is set to **TRUE.** The output format will be discussed in class and provided as an addendum after the class.

- FORMAT: **krushkal**
  - EXAMPLE: **krushkal**
    - Return: **Min Spanning Tree as an ADJACENCY LIST** (just call your print function)
    - Error: Edges are deleted directionally, so if even ONE edge was deleted (UNDIRECTED = FALSE), you should assume that Krushkal's algorithm can no longer be executed on the graph. Print the following**: ERROR: Krushkal's Algorithm available only for UNDIRECTED graphs**

7. **Print the Graph in Adjacency Matrix Format.**
   - FORMAT: **print matrix ***** ALPHABETIC ORDER by Node Names on axes *****
8. **Print the Graph in Adjacency List Format.**
   - FORMAT: **print list ***** Print in ALPHABETIC ORDER by Node Names *****
9. **Execute command from a Command File.** Commands can be stored in a CSV file to be exectuted one at a time, just like they were being input from the command line. The format of the commands is identical to the descriptions above EXCEPT that the individual fields are separated by commas. Note that it is also possible that there may be extra commas at the end of some or ALL of the instruction lines. Your program must be able to tolerate (ignore) these extraneous commas at the end of the lines.
   - FORMAT: **file \home\pwb0013\csce2110\program2\data\Prog2_Test.csv**
10. **Quit execution.**
   - FORMAT: **quit**

PROGRAM SUBMISSION:

Submit INDIVIDUAL program files with CSCE-2100 Specified Filenames (**NO *.zip files, NO *.tar files, NO *.rar files**) to BlackBoard before the due date. The file naming convention must conform to the standard as described in the instructions for Program 1. Specifically:

LxxF_10_P3_yyy.cpp -OR- *.h

where: "Lxx" is the first 3 character of your LAST name.
"F" is the first character of your FIRST name.
"10" identifies the assignment is for CSCE2110.
"P3" is the identifier for this assignment (P3=Program 3).
"yyy" is any other text (descriptive name) that you choose as your unique filename(s).

EXAMPLE: **BurP_10_P3_Main.cpp**

Multiple submissions of one or all of your files will be allowed. Your program files will all be extracted to a single UNIX (csexx.cse.unt.edu) subfolder and will be compiled with the command: **make**
Once compiled, we will execute with command: **make run**

You are allowed AND ENCOURAGED to discuss the details of this assignment with your Teammates. Programs 2-4 will be submitted by each person. Program 4 will be a single submission for the entire Team. You should be working together so that each individual understands exactly how this program works.I also encourage the reuse of code, even from books or the internet. You MUST specify the source if you are reusing code, but why reinvent the wheel? Instead, start with something that works ... that YOU UNDERSTAND ... and spend your time modifying that code to do your bidding! That is not cheating ... that is working smart.

We will be discussing the overall project in class throughout the semester. Use these discussions to shape your code into something that can be reusable with minimal effort for the follow-on functionality. Remember to use functions to do most of the work to isolate changes in input/output/storage to single functions, instead of having to rewrite all of the code. Failure to submit to BlackBoard ON-TIME will result in a score of ZERO for the assignment.