**Program 3 - Graphs**

[My Solutions](https://vpn1.unt.edu/+CSCO+dh756767633A2F2F63656E7867627A6E672E7066722E6861672E7271683A3231313038++/tasks/51/solutionupload/)

**(75 points)**

Implement a weighted directed graph internally represented by an adjacency list. The graph will contain a fixed number of nodes. Use an array to represent these “buckets”.  Lists will grow out of each bucket to represent edges. You should have a class for your graph, for the list implementation and for the nodes. **You are not allowed to use the STL.**

Once the program is started, it will print out the promt “graph> ” (**> is followed by a whitespace**):

./a.out

graph>

**You will implement the commands “create”, “insert”, “remove”, “print” and “quit”:**

**create**

Create takes a single argument, representing the number of nodes. Keep in mind that the number of nodes determines the number of buckets in your array. Create a graph that does not contain any edges. If you already have created a graph, it will be deleted and replaced by the new one. Then repeat the prompt.

graph> create 4   
graph>

**insert**

Insert takes 3 arguments: the source node, the destination node and the weight of the edge. If and edge already exists, replace the weight by the new weight. Then repeat the prompt.

graph> insert 1 3 7  
graph> insert 3 0 4  
graph> 

**remove**

Remove takes 2 arguments: The source node and the destination node. Remove the node. Then repeat the prompt.  

graph> remove 1 3   
graph> 

**print**

Print takes no argument. Print out all the lists in the concatenated on a single line. Start with bucket 0. The format for a single entry is (node1, node2, weight). Do NOT preprend the prompt to the pintout. Then repeat the prompt.

graph> print  
(0,2,4)(0,1,3)(1,3,6)(3,0,4)  
graph> 

**quit**

Exit the program

graph> quit

**Error Handling**

* If the command received from the user input is not supported, print out an error message starting with “Error!”.  (Do not capitalize the entire word “Error”)
* If the user tries to add an edge connected to a non-existing node, print out an error message starting with “Error!”.  (Do not capitalize the entire word “Error”)
* If the user tries to remove an edge that does not exist, print out an error message starting with “Error!”.  (Do not capitalize the entire word “Error”)

**Submit AT LEAST the following files:**

* Your main file controling the flow of the program
* The prototype files for your graph, list, and node classes.
* The implementation files for your graph, list, and node classes.

**Example of program execution:**

g++ \*.cpp  
./a.out  
  
graph> create 4  
graph> insert 1 3 7  
graph> insert 3 0 4  
graph> remove 1 3   
graph> print  
(3,0,4)  
graph> insert 1 3 6  
graph> insert 0 2 4  
graph> print  
(0,2,4)(1,3,6)(3,0,4)  
graph> insert 0 1 3  
graph> print  
(0,2,4)(0,1,3)(1,3,6)(3,0,4)  
graph> hi  
Error! Command not supported.  
graph> insert 3 7 2  
Error! Node does not exist!  
graph> remove 2 0  
Error! Edge does not exist!  
graph> remove 7 2  
Error! Node does not exist!  
graph> insert 0 3 7  
graph> print  
(0,2,4)(0,1,3)(0,3,7)(1,3,6)(3,0,4)  
graph> quit

**Your program will be judged on the following:**

* 35 points - Passes I/O requirements
* 35 points - Code satisfies requirements of assignment
* 5 points - Professional coding style  
  - 2 Adequate comments  
  - 2 Modularity (small main function, separate functions, etc)  
  - 1 Readability (line length, indentation, variable names)