Assignment Objective: Build skills on C class creation and integration while implementing a **diGraph** management and analysis system, for directed graphs.

Requirements:

* Create the graph class with the following variables and functions within graph.h and graph.cpp.
  + Public functions:
    - graph(int vidCapacity, bool directed); // sets up the empty graph
      * Allocates “a”, an array of size vidCapacity pointers that each point to an array of size vidCapacity, and vertexList, an vidCapaicity-sized array of type vertex (see below). Sets vCount and eCount to 0.
      * Sets the member variable “directed” accordingly. For this assignment, it shall be presumed to be set to true in all graph instantiations.
    - ~graph(); // deletes the array “a”
    - bool createV(int label); // creates the node labeled by the parameter; returns true if label not already used and there is a label table entry left to be allocated to the label; returns false otherwise. I.e., this requires vCount < vidCapacity to allocate a node to a label. If all went well, then vertexList[vCount].label was set to “label”, and vCount was then incremented by one.
    - bool addEdge(int uLabel, int vLabel, int weight); // creates the edge with the given weight. Weight must be > 0. If the edge is added, eCount is incremented, and true is returned. It causes the nodes for the passed labels to be created if they don’t already exist. Returns false if nodes can’t be created or if the weight is not a positive, non-zero integer. Returns false if the edge had already been created; i.e., can’t change the weight of edge this way.
    - bool deleteEdge(int uLabel, int vLabel); // deletes the edge, if the edge had been set and decrements eCount; returns true if it deletes it; returns false if the edge was already deleted; returns false if the labels didn’t match existing nodes.
    - Int getEdgeWeight(int uLabel, int vLabel); // returns the weight of an edge; returns 0 if the edge does not exist; returns -1 if either of the labels do not exist.
    - void clear(); // causes the graph to be reset to its original state, with no vertices or edges
    - int inDegree(int label) const; // return inD of the node; -1 if the node does not exist
    - int outDegree(int label) const; // return outD of the node; -1 if the node does not exist
    - int sizeV() const; // returns the largest size the graph could be, vidCapacity.
    - int sizeUsedV() const; // returns the number of nodes actually used, vCount
    - int sizeE() const; // returns the number of edges in the graph, eCount
    - void printIt() const; // prints the graph; see sample output for proper format
    - void printAllPaths(); // placeholder for the next assignment.
    - **NOTE: Ensure printIt() matches the output of the diGraphCorrectOutput.txt file**
  + Private functions:
    - int min(int x, int y) const; // returns the minimum of x and y
    - int max(int x, int y) const; // returns the maximum of x and y
    - int vidToLabel(int vid) const; // Returns the label associated with vertex vid, or -1 if the vid is not in range.
    - int labelToVid(int ulabel) const; // returns the vid for the given label, otherwise -1 if the label is not in the vertexList.
    - bool isEdge(int uLabel, int vLabel) const; // returns true if the edge exists.
    - bool isV(int label) const; // returns true if there is a vid associated with the given label
  + Private variables and declarations:

enum color { WHITE, GRAY, BLACK };

class vertex {

public:

int label; // what is the world's name for this vertex

int color; // for BFS

int distance; // for BFS and DFS

int predVID; //

int startTime; // for DFS

int finishTime; // for DFS

};

* + - int \*\*a; // the array for the graph; size will be vidCapacity
    - vertex \*vertexList; // an array of vertex type; size will be vidCapacity
    - bool directed; // memorializes whether the graph is directed; defaults to true.
    - int vidCapacity; // memorializes the dimensions of the graph matrix
    - int vCount; // tracks the number of nodes in the graph
    - int eCount; // tracks the number of edges in the graph
* The vertexList is an array that stores various attributes of a vertex (AKA node), quite notably the label of the given vertex and attributes that are set during the various traversal algorithms.
* You must not use any other data structure, whether built-in or otherwise.
* Make a Makefile that builds graph.o and graphMain.o, and then graph from those two; quite similar to prior Makefiles. Be sure to use the -O2 compiler optimization and the standard CWFLAGS that we have been using.
* Demonstrate that the graph structure works:
  + Compile your program as follows:

make

* + Run your program as follows:

graph < graphInput.txt > graphOutput.txt

* + Compare your output file, graphOutput.txt, to the posted graphCorrectOutput.txt file.
* Deliverables:
  + Into D2L put a zip file containing:
    - The Makefile
    - A graph.h file for your graph ADT
    - A graph.cpp file for your graph ADT implementation
    - A graphOutput.txt text file with your output
    - DO NOT CHANGE THE NAMES OF THE FILES
    - DO NOT put a project into D2L
  + Turned into class: a hardcopy of the files above.