Assignment Objective: Demonstrate the course skills while implementing a graph ADT that supports directed and undirected graphs, performs a depth first search, and determines whether a graph (directed or undirected) is cyclic.

Requirements:

* Copy the final version of the diGraph BFS files, graph.cpp and graph.h, along with the required ancillary .cpp and .h files.
* Within the graph.h and graph.cpp files, do the following:
  + Private variable:
    - Add “bool cycleFound” to be used in the isCyclic() set of functions (described below).
  + Private functions:
    - Create void dfs(int uVid, int &timestamp) to do the private part of the DFS algorithm as described in CLRS p 565.
    - Create void dfs(int vVid) that does a dfs, but only starting at v, not all the vertices as does the above dfs().
    - Create bool isCyclicDirected() that determines whether a directed graph is cyclic.
    - Create bool isCyclicUndirected() that determines whether an undirected graph is cyclic.
  + Public functions:
    - Ensure the constructor captures the setting of “directed”.
    - Modify “bool deleteEdge(int uLabel, int vLabel)” to be “bool deleteEdge(int uLabel, int vLabel, &weight)” such that the weight of a successfully deleted edge is given back to the caller through the reference variable “weight”. The weight should be set to -1 if the edge is not deleted. For an undirected graph, delete the “reflective” edge as well. Do not make any further changes to deleteEdge().
    - Modify addEdge() to add the reflective edge for an undirected graph. Don’t double-count the edges.
    - Modify inDegree() and outDegree() to return -1 if the graph is not directed.
    - Add a new public “int degree(int label)” to return the degree of a vertex in an undirected graph. If label is not an existing vertex or the graph is not undirected, it shall return -1.
    - Create a new “void dfs()” that implements the DFS algorithm as described in CLRS p 565.
    - Modify printIt() to have the same output as the ugraphCorrectOutput files.
      * #include <iomanip> and use setw(2) as appropriate
    - Create a bool isCyclic() that returns the results of its calling isCyclicDirected() or isCyclicUndirected, as appropriate.
* Note: An edge (v, v) is known as a self-loop. In a directed graph, a self-loop constitutes a cycle. In an undirected graph, a self-loop does not constitute a cycle.
* You must not use any other data structure, whether built-in or otherwise.
* **Demonstrate your code works by doing the following:**
  + Compile your program with a Makefile using graphMain.cpp, graph.cpp, queue.cpp, cList.cpp and their associated header files.
  + Run your program as follows:

./graph 0 < ugraphInput1.txt > ugraphOutput1false.txt

./graph 0 < ugraphInput2.txt > ugraphOutput2false.txt

./graph 1 < ugraphInput1.txt > ugraphOutput1true.txt

./graph 1 < ugraphInput2.txt > ugraphOutput2true.txt

* + Compare each output to the Correct version.
* **Deliverables:**
  + Zip the following files into a single zip file:
    - cList.h, cList.cpp, queue.h, queue.cpp, graph.h, graph.cpp, your four output files, and your Makefile
    - DO NOT CHANGE THE NAMES OF THE FILES
    - DO NOT put a project into D2L
  + Turned into class: a hardcopy of the same files, in that order, but not the Makefile.