Assignment Objective: Demonstrate the course skills while implementing a graph ADT that supports directed and undirected graphs.

Requirements:

* Copy the final version of p10.cpp and p10.h to p11.cpp and p11.h
* Within the p11 files, make the following changes:
  + Private functions:
    - Convert ind(int x, int y) to check for whether the graph is undirected or directed. If directed, return x\*n + y. Otherwise, if undirected, if x < y, return x\*n + y, else return y\*n + x.
    - Create bool isCyclicDirected() that determines whether a directed graph is cyclic. It need not verify to see if the graph is directed.
    - Create bool isCyclicUndirected() that determines whether an undirected graph is cyclic. It need not verify to see if the graph is undirected.
  + Public functions:
    - Convert 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”. If the edge is not deleted, the weight should be set to -1. Do not make any further changes to deleteEdge().
    - Do NOT change addEdge().
    - 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.
    - Modify printIt() to have the same output as the p11CorrectOutput files.
    - Create a bool isCyclic() that returns the results of its calling isCyclicDreicted() or isCyclicUndirected, as appropriate.
    - Ensure that isPath(vLabel, wLabel) works properly if v and w are the same label.
* Hint: to match the correct output, in p11.cpp add an include of <iomanip> and use the setw() function where needed.
* Note: An edge (v, v) is known as a loop. It is an edge from v to v, known as a loop. in a directed graph, a loop constitutes a cycle. In an undirected graph, a 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 as follows:

g++ p1.cpp cil.cpp iq.cpp p11.cpp p11m.cpp -o p11

* + Run your program as follows:

p11 0 < p11input1.txt > p11output1false.txt

p11 0 < p11input2.txt > p11output2false.txt

p11 1 < p11input1.txt > p11output1true.txt

p11 1 < p11input2.txt > p11output1true.txt

* **Deliverables:**
  + Zip the following files into a single zip file:
    - A p11.h file for your updated graph ADT
    - A p11.cpp file for your updated graph ADT implementation
    - All of your output files from above.
    - DO NOT CHANGE THE NAMES OF THE FILES
    - DO NOT put a project into D2L
  + Turned into class: a hardcopy of p11.h, p11.cpp, p11output1false.txt, in that order.