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

Requirements:

* Create the **Graph** class that works with an intList derived from your P1. **Graph** will have the following variables and functions:
  + Public method:
    - Graph(int n = 100, bool directed = true); // sets up the empty graph
      * Allocates “a”, an n\*n sized array, and labels, an n-sized array. Sets all values of a to zero. Allocates the labels intList. Sets vCount and eCount to 0. And more.
    - ~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 < n to allocate a node to a label. If all went well, then labels[latestIndex] gets set to “label”, where latestIndex is the new vCount - 1.
    - 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.
    - void clear(); // causes the graph to be reset to its original state, with no vertices or edges
    - bool isEdge(int uLabel, int vLabel) const; // returns true if the edge exists; returns false if the node labels are not both assigned to a node or if the edge does not exist (weight == 0).
    - bool isV(int label) const; // return true if there is a node labeled by label
    - 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 (n).
    - int sizeUsedV() const; // returns the number of nodes actually used, vCount
    - int sizeE() const; // returns the number of edges in the graph, eCount
    - printIt() const; // prints the graph; see sample output for proper format
    - **NOTE: Change printIt() to match the output of the p9aCorrectOutput.txt file**
  + Private functions:
    - int ind(int x, int y) const; // returns the mapping of x,y to index; uses least vertex id first for undirected graphs
    - int labelToVid(int label) const; // returns the vertex index for the given label; returns -1 if no node has that label
  + Private variable:
    - int \*a; // the array for the graph; size will be n\*n
    - intList \*labels; // stores the labels of the edge; size will be n
    - int n; // 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
    - bool directed; // remembers whether the graph is directed or not
* Non-member functions:
  + int min(int x, int y); // returns the minimum of x and y
  + int max(int x, int y); // returns the maximum of x and y
* The labels array is a list that maps “label” to a node/vertex number. Node numbers are allocated from 0 to n-1, as nodes are created. Think of it as a lookup table.
  + Use labels->add(label) to add a new vertex to the Graph
  + Use labels->find(label) to get the vertex/node number of the vertex with that label; it will return -1 if the labeled vertex is not in the graph.
  + Use labels->readAt(index) to get the label stored at that vertex. readAt is new an different than before; it’s prototype is “int readAt(int index)”; it returns -1 if the index is invalid; it returns the label at that index otherwise.
* You must not use any other data structure, whether built-in or otherwise.
* Demonstrate that the graph structure works:
  + Compile your program as follows:

g++ p1.cpp p9a.cpp p9am.cpp -o p9a

* + Run your program as follows:

p9a < p9aInput.txt > p9aOutput.txt

* + Compare your output file, p9aOutput.txt, to the posted p9aCorrectOutput.txt file
* Deliverables:
  + Into D2L put a zip file containing:
    - A p1.h file for your intList ADT
    - A p1.cpp file for your intList ADT implementation
    - A p9a.h file for your graph ADT
    - A p9a.cpp file for your graph ADT implementation
    - A p9aOutput.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.