CSCI 340 **Computer Assignment 9** **Spring 2019**
(10 points)

Simple graph

For this computer assignment, you are to write a C++ program to implement several graph algorithms on simple graphs. These graphs are implemented with adjacency list representation. The `graph` class is already defined in the header file `assignment9.h` in directory: `/home/turing/mhou/public/csci340spring2019.` This directory also contains all other files related to this assignment.

The graph is created from a text file that contains the adjacency matrix representation of a graph. The first line of the file is an integer `n` indicating the number of vertices in the graph. Next there is a `(n+1)×(n+1)` table where the first row and the first column are names of vertices. The table records edges of the graph. Integer `1` indicates an edge exists between a pair of vertices. Integer `0` indicates no edge.

For a given graph `size,` each vertex of a graph can be referenced by index values `0, 1, …, ( size – 1 );` however, vertices of a graph are labeled consecutively by capital letters, starting from `A,` which corresponds to the index value `0.` Use index values when you refer to a vertex in a graph in implementing the algorithms; use vertex labels only in printing. The edges are recorded in the data member `adj_list`, which is a `vector` object. The vertex labels are recorded in the data member `labels`, which is also a `vector` object. For example, you can use `labels[0]` to get the name of the first vertex, and `adj_list[0]` to get the `list` of edges (recorded as integer indexes of destination vertices) from the first vertex as the source.

The source file assignment9.cc already contains the complete main function. Implement **ALL** member functions of the `graph` class and put your implementations of these functions in this source file. (You can certainly insert inline code in the header file when the implementation of a member function only involves **at most** of couple of lines of code.) **Only several** functions are described in more details below.

- `graph :: graph ( const char* filename )` : This is the constructor. It reads from the input file of the graph with adjacency matrix representation and builds the graph with adjacency list representation. This method sets the value of `size`, builds the vectors `labels` and `adj_list`. For example, for the following line of input:
  D 0 1 0 0 1 0 0
  Add edges to `adj_list[3]`, which records edges starting from vertex `D`, by adding values 1 and 4, which are indexes for vertices `B` and `E`.

- `void graph :: depth_first ( int v )` : This private function is used to traverse a graph in the *depth-first traversal/search* algorithm starting at the vertex with the index value of `v`. To implement this method (and together with the `traverse` method below), you may need several global variable and objects. E.g. container objects to record the visiting order of all vertices, the container object to record the paths of traversing edges, and an integer indicating the current order in traversing.

- `void graph :: traverse ( )` : This public function is used to traverse a graph and invokes the above `depth_first` method. You will also need to display traverse result: the list of vertices in the order of their visit and the list of edges showing the path(s) of the traversal. At beginning of this method, you need to initialize the global variable(s) and object(s) used in `depth_first`.

- `void graph :: print ( ) const` : This function prints the adjacency list for the graph. The following line is an example from an output.

  D: B, E,

  It indicates there are edges from vertex `D` to vertices `B` and `E`.

Programming Note:

- Include any necessary headers and add necessary global variables and objects.

- You are not allowed to use any I/O functions from the C library, such as `scanf` or `printf.` Instead, use the I/O functions from the C++ library, such as `cin` or `cout.`

- In the final version of your assignment, you are not supposed to change existing code, including class definition and the `main` method, provided to you in the original files `assignment9.h` and `assginment9.cc`. You can insert new code to both files, including data members to the header file if necessary. Usually the source file (.cc) contains the implementation of member methods of the class. You can add inline code in the header file (.h) when the implementation is very brief, containing just one or two lines for a member method.

- To compile the source file, execute "`g++ -Wall assignment9.cc –o assignment9.exe`". This will create the executable file `assignment9.exe.` To test your program, execute "./`assignment9.exe assignment9input.txt &> assignment9.out`", which will put the output (including any error messages) in file `assignment9.out`. You can find the correct output of this program in file `assignment9.out` in the directory shown in the last page. `assignment9input.txt` is also in that directory.

- Add documentation to your source file.

- Prepare your `Makefile` so that the TA only needs to invoke the command "`make`" to compile your source file and produce the executable file `assignment9.exe`. Make sure you use exactly the same file names specified here, i.e. `assignment9.cc` and `assignment9.exe`, in your `Makefile`. Otherwise your submission will get 0 points.

- When your program is ready, submit your files `assignment9.h`, `assignment9.cc` and `Makefile` to your TA by following the Assignment Submission Instructions.