CS 3100/5100
Programming Project 4

A strongly connected component in a directed graph DG=<V,E> is a set of vertices C (subset of V) such that for every pair of vertices x, y in C there is a directed path from x to y and a directed path from y to x. Thus, starting at any vertex in a strongly connected component, it is possible to reach every other vertex in the component.

The algorithm for finding strong components is relatively simple.

1. Perform a depth first search on the directed graph (DG).

2. Number the depth first search (DFS) tree (or DFS forest) using a postorder traversal.

3. Form a new graph DG_R=<V, E_R> by reversing every edge in E to form E_R.

4. Perform a depth first search on the directed graph DG_R. Make sure to always start the search at the vertex with the highest DFS number generated in step 2. If some of the vertices are not processed, restart the search at the highest DFS numbered vertex that is not marked as visited.

Each DFS tree produced in step 4 contains a set of vertices that define a strongly connected component in the original graph. Notice that every vertex is in a connected component even if there are no edges entering or exiting the vertex.

Your task is to define a directed graph class to:

1. read a graph from the keyboard
2. print an adjacency matrix or list showing the content of the original graph
3. perform the depth first search
4. display the DFS forest with labels identifying each tree
5. form the reverse graph
6. print an adjacency matrix or list showing the content of the reverse graph
7. display the DFS forest for the reverse graph showing the strongly connected components

You may assume that graphs will contain no more than 50 vertices. A graph will consist of a ordered sequence of vertex labels in the range of 0-49 that will start with 0. The input will consist of a single integer indicating how many vertices are used in the graph followed by pairs of integers that represent directed edges. **Do not prompt the user to enter data.** You may assume the input data is correct. Sample input is shown below:

```
6                       <=== 6 vertices in the graph numbered 0, 1, 2, 3, 4, 5
1 4                     <=== edge <1,4>
3 4                     <=== edge <3,4>
0 1
4 5
5 1
<EOF>                   <=== end of file
```

Your directed graph class should be simple and easy to understand. The main program should define a directed graph variable and then perform a series of calls to accomplish the tasks (1-7) outlined above. You do not have to define separate functions for each of the tasks, but generic tasks such as DFS should be initiated by a call to public member function. At least 20% of your project grade will be determined by the quality of the design and implementation of directed graph class. Documentation is required and worth a minimum of 10% of the project grade. All output should be carefully formatted and labeled. On the assigned due date, submit all source files (*.cpp and *.h) used in your solution along with a makefile that can be used to build your application. **The target executable file must be named project4.** I will test your program by issuing the commands:

```
make
project4 < graph.dat
```

where graph.dat data contains the vertex numbers defining the edges.