CSS 343 Assignment #3 Due: Wednesday, Apr 30, 2014 Spring 2014
(Turn in instructions on website, assignments page.)

## **Part 2, Programming** (depth-first search)

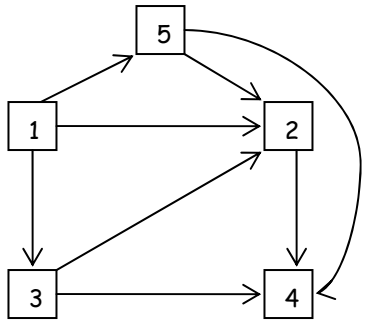
Display the graph information and implement depth-first search (using the ordering as given by the data, i.e., start at one). Display the node numbers in depth-first order. Implement the specified graph ADT functions.

In the data, the first line tells the number of nodes, say n. Following is a text description of each of the 1 through n nodes, one description per line (max length of 50). After that, each line consists of 2 ints representing an edge. If there is an edge from node 1 to node 2, the data is: `1 2` . A zero for the first integer signifies the end of the data for that one graph. All the edges for the first node will be together first, then all the edges for the second node, etc. Take them as they come, no sorting. (Edges in input will always be in reverse order in the output, discussed below.) There are several graphs, each having at most 100 nodes. Assume the input data file has correctly formatted data. For example:

Sample Input:

```
5
Aurora and 85th
Green Lake Starbucks
Woodland Park Zoo
Troll under bridge
PCC
1 5
1 3
1 2
2 4
3 4
3 2
5 4
5 2
0 0
```

picture (not part of data):




Sample Output

```
Graph:
Node 1       Aurora and 85th
  edge 1 2
  edge 1 3
  edge 1 5
Node 2       Green Lake Starbucks
  edge 2 4
Node 3       Woodland Park Zoo
  edge 3 2
  edge 3 4
Node 4       Troll under bridge
Node 5       PCC
  edge 5 2
  edge 5 4

Depth-first ordering: 1 2 4 3 5
```

## **Part 2 Notes**

-- Implement using an adjacency list (array of lists). Add any needed data, e.g., a field in the graph node can be used to mark visiting a node. The object with the information on a graph node can either be directly stored in GraphNode (as in NodeData) or as a pointer to an object (like NodeData*). As with part 1, start in array element one.

```
struct EdgeNode;          // forward reference for the compiler
struct GraphNode {        // structs used for simplicity, use classes if desired
   EdgeNode* edgeHead;    // head of the list of edges
   ...
};

struct EdgeNode {
   int adjGraphNode;      // subscript of the adjacent graph node
   EdgeNode* nextEdge;
};

class GraphL {
public:
   ...
private:
   // array of GraphNodes
};
```

-- You do not need to implement a complete Graph class. Normally you would have a copy constructor, etc., but you don't need to implement all of them if you promise not to do that in real life. You will implement the constructor, destructor, buildGraph, displayGraph, and depthFirstSearch. A driver that (partially) tests your code is on the web site.

-- To simplify things, you should always insert EdgeNodes at the beginning of the adjacency list for a Node. Your output of the edges for each Node will, thus, be in the reverse order in which they are listed in the input file (see the example above.) An EdgeNode list may not be sorted. (They are sorted in the example for readability.) Make sure to follow this coding simplification (and process the edges in the order they are in the list,) since it affects the depth-first ordering that you will get. Note that you must handle a graph with disconnected components.