CSCE 221 Cover Page
Programming Assignment #3

First Name　　　　Last Name　　　　UIN

User Name　　　　E-mail address

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero. According to the University Regulations, Section 42, scholastic dishonesty are including: acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion read more: Aggie Honor System Office

Type of sources			
People			
Web pages (provide URL)			
Printed material			
Other Sources			

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work."

Electronic signature　　　　Date

Programming Assignment 3 (100 points) – due June 27

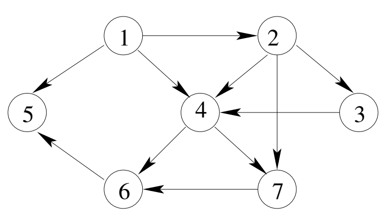
Consider a directed graph without cycles called **a directed acyclic graph** (DAG). In this assignment you are going to find a topological ordering in a DAG. There are many real life problems that can be modeled by such graphs and solved by the topological ordering algorithm. Read the section 9.2, pp. 382-385 in the textbook to learn more about the algorithm.

- The assignment consists of three parts:
  - **Part 1** – implementation of the graph data structure
  - **Part 2** – implementation of the topological ordering for a DAG. Please notice that we take a DAG as an input and the topological ordering for the DAG is returned; or the exception message is displayed: "*There are cycles in the graph*".
  - **Part 3** – preparing a report:
    - ∗ discussing the implementation of the Part 1 and 2 and the running time of the algorithms used to solve the problem.
    - ∗ providing testing cases for correctness
- **Part 1 (40 points)** – demo to your TA in the first lab of June 20 week.

  In this part you should implement a graph data structure which is defined based on an additional type `Vertex`. You can download the supplementary (zipped) file with a code skeleton from the eCampus. The implementation of the `Graph` class should be based on adjacency lists, see the file `graph.h`.

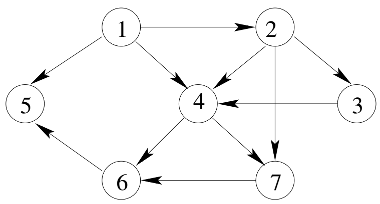
  The graph is populated by reading data from a text file with fixed format, see the example below. At each row, the first number is the label of the start vertex of a directed edge. Other numbers in this row are the end vertices accessed from the start vertex.

  Example. The first row starts with the vertex 1 and provides information about three directed edges to vertices 2, 4 and 5. The number -1 is used as a terminator of a line. In the case when there is no edge for a certain vertex, for example for the vertex 5, the list is empty. This input file is called `input.data` in the supplementary file.




```
1 2 4 5 -1
2 3 4 7 -1
3 4 -1
4 6 7 -1
5 -1
6 5 -1
7 6 -1
```

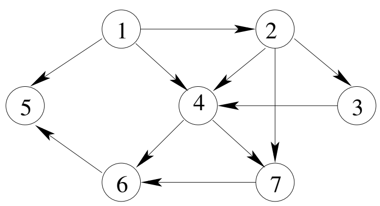
  - The purpose of this part is to read in the data from an input file with a given format, build a graph data structure, and display the graph on the screen in text format.
  - We assume that the graph we are dealing with is sparse and unweighted. Then, adjacency lists will be a natural choice to store the connection between two nodes. The class `Graph` is used to store the graph and implements the necessary operations such as `buildGraph`, and so on. Furthermore, a `Vertex` class can be implemented to store the basic information about a graph node such as a label which in our case is an integer.
  - **We assume that the graph nodes are numbered consecutively starting from 1, and there are no gaps in the node numbering.**
  - `displayGraph()` should print out each vertex and its adjacency list on the screen. For example, consider the graph *G* and its corresponding adjacency linked lists for an input sample graph (`input.data`). Test your program by reading a graph from an input file and use the function `displayGraph()` to display the generated graph in text format on the screen, see the format of the output below.



1 : 2 4 5
2 : 3 4 7
3 : 4
4 : 6 7
5 :
6 : 5
7 : 6

- You can compile your code using this command line:
  `make`
- And you can run your program by executing:
  `./main input.data`

- **Part 2 (40 points)** – due Jun 27 to eCampus.
  - The formal definition of a topological sort:
    Let G be a DAG with n vertices. A **topological ordering** of G is the ordering $v_1, v_2, \ldots, v_n$ of the vertices of G such that for every edge (v_i, v_j) of G, $i < j$.
  - The illustration of the definition of the topological sort ordering gives a sequence of vertices:



`1 2 3 4 7 6 5`

    The topological sort ordering places vertices of the graph along the horizontal line with the following property: if there is an edge from the vertex v_i to the vertex v_j then the vertex v_i precedes v_j in the topological ordering.
  - Topological sort algorithm:
    1. The input is a DAG
    2. Algorithm – see the textbook, Fig. 9.7, p. 385.
       * You can use `topNum` (`top_num`) as in Fig. 9.7, and then traverse the graph to initialize the topological sort ordering vector.
    3. The output of the program should be a vector of vertices (or their labels) set in topological sort order.
       * You need to print the topological sort ordering vector by printing the labels of vertices.

- **Part 3 (20 points) – due April 27**
  - Submit to eCampus: the source code and an electronic version of your report including
    * (15 points) description of your implementation, C++ features used, assumptions on input data.
      · Why does the algorithm use a queue? Can we use a stack instead?
      · Can you explain why the algorithm detects cycles?
      · What is the running time for each function? Use the Big-O notation asymptotic notation and justify your answer.
    * (5 points) test your program for correctness using the four cases below:
      Case 1: Use the example (`input.data`) provided in the description of the problem.
      Case 2: Samantha plans her course schedule. She is interested in the following eight courses: CSCE121, CSCE222, CSCE221, CSCE312, CSCE314, CSCE313, CSCE315, and CSCE411. The course prerequisites are:

course	#	prerequisites	
CSCE121:	1	(none)	
CSCE222:	2	(none)	
CSCE221:	3	CSCE121	CSCE222
CSCE312:	4	CSCE221	
CSCE314:	5	CSCE221	
CSCE313:	6	CSCE221	
CSCE315:	7	CSCE312	CSCE314
CSCE411:	8	CSCE222	CSCE221

Find a sequence of courses that allows Samantha to satisfy all the prerequisites. Assume that she can only take one class at a time. The input file for this case is provided (`input2.data`)

Case 3: Samantha loves foreign languages and wants to plan her course schedule. She is interested in the following nine courses: LA15, LA16, LA22, LA31, LA32, LA126, LA127, LA141, and LA169. The course prerequisite are:

course	#	prerequisites	
LA15:	1	(none)	
LA16:	2	LA15	
LA22:	3	(none)	
LA31:	4	LA15	
LA32:	5	LA16	LA31
LA126:	6	LA22	LA32
LA127:	7	LA16	
LA141:	8	LA22	LA16
LA169:	9	LA32	

Find a sequence of courses that allows Samantha to satisfy all the prerequisites. Assume that she can only take one class at a time.

Case 4. Create a directed graph with cycles and test your program. There is one such a file provided (`input-cycle.data`).