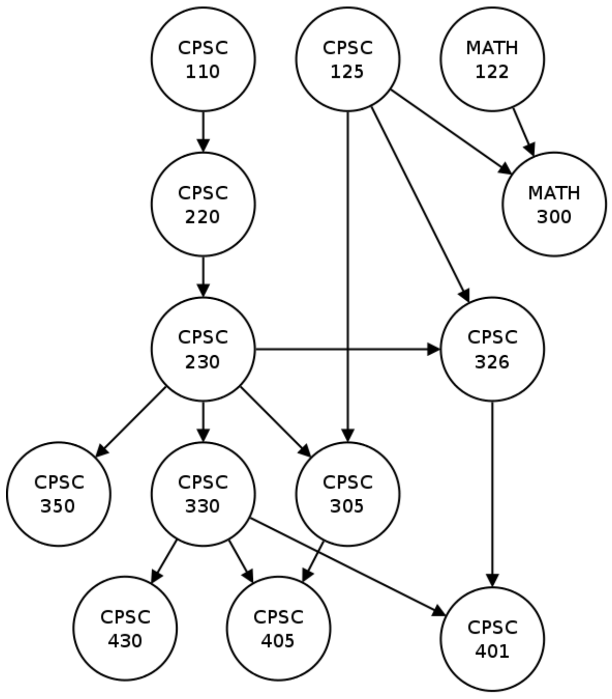
cs230 - class scheduler

assignment written by Professor Finlayson

Task

Directed Acyclic Graphs (DAGs) are an important type of graph with many applications. Recall that a directed graph is one where the edges have a direction. Recall that an acyclic graph is one with no cycles. A cycle is a sequence of edges in a graph where following it will lead you back to the original node. If a graph is directed and acyclic, it is a DAG.

One common usage of DAGs is to represent dependencies between different things. For example, courses in a school generally can have prerequisites. In the computer science department, taking CPSC 230 depends on having taken CPSC 220. We can represent these prerequisites with a DAG where an edge from course A to course B means that A is a prerequisite of B. Below is a DAG representing the dependencies between several required courses for the computer science (traditional track) major:



Your task

Your task is to develop a scheduler for a student so they graduate in the minimum number of semesters. Your program should take 2 command line arguments: the maximum number of computer science courses the student would like to take per semester and a filename. The file consists of a list of courses with their pre-requisites in the following format:

```
course1 number-of-prereqs prereq1 prereq2 ...
course2 number-of-prereqs prereq1 prereq2 ...
...
```

Each course will be specified with four letters followed by 3 digits.
Some courses will of course have no pre-requisites. There will be no more than 100 courses total. Here is an input file representing the CS courses given above. Here is an input file with a cycle.

If a student is willing to take as many computer science courses as possible per semester the command line argument for max courses will be -1.

The task Professor Finlayson's students did was slightly different. It was to topologically sort the courses.

One algorithm for topological sorting is given below:

```
1. Set the ordering to empty.
2. Find the set of nodes with no edges coming into them. Call this the active set.
3. While there are nodes in the active set:
    1. Move a node N from the active set to the ordering.
    2. For each edge coming out of N and into M:
        1. Remove the edge from the graph.
        2. If M now has no edges coming into it, add it to the active set.
4. If the graph has any edges left, then there is no topological ordering!
5. Otherwise, the topological ordering is in the "ordering" list.
```

The output should look like

```
Semester 1: CPSC110, MATH122, CPSC125
Semester 2: CPSC220, MATH300
Semester 3: CPSC230
Semester 4: CPSC305, CPSC326, CPSC330, CPSC350
Semester 5: CPSC405, CPSC401, CPSC430
```

Details

- You can have whatever files, classes and functions you want for this project.
- You can base your graph off of an adjacency matrix or an incidence list.
- The file name for input will be given on the command line (you should also check for errors).
- You should then read the courses into a graph setting up the nodes and edges appropriately.
- It may help to make two passes over the file: one to add the nodes and one to add the edges. To start back at the beginning of a file, you can call the following two functions (where "file" is your ifstream object):

-
  ```
  file.clear( );              // reset end of file to false
  file.seekg(0, ios::beg);    // move the input back to the start
  ```

XP and submission

It is easier to implementing the algorithm that handles the cases when the number of courses a person wants to take is 1 or as many as possible. You will get 50XP when you implement one of these and 25XP more if you implement the other. You will get an additional 50XP if your algorithm handles cases like when a person wants to take 2 computer science courses per semester.

Please submit your code to submit.o.bot

DUE: 9 November