It’s time to register for courses. This assignment will help you plan out your schedule of computer science courses for the next few years. First you will create data structures that represent the collection of all computer science courses and the prerequisites of each course. Then you will implement an algorithm that performs a topological sort of these courses. Similar to a regular sorting method, a topological sorting method takes a collection of objects and puts them in order according to a ordering criterion; however, topological sorting works with collections of data that are partially ordered, i.e., some pairs of data are comparable to each other, while other pairs of data are not comparable. For example, computer science 101 is a prerequisite for computer science 102, so those two courses are comparable: a student must take 101 before 102. On the other hand, computer science 102 and 145 are not comparable because neither one is a prerequisite for the other. A student may take them in any order. Notice that 101 is an indirect prerequisite for 224, since 101 must be taken before 102 and 102 must be taken before 224. (A similar argument applies using 101, 145 and 224).

When topological sorting is applied to the collection of computer science courses (partially ordered by the prerequisite relation between courses) the result is an ordering of courses that is consistent with the prerequisites. When topological sorting is applied to the diagram above, there are two possible solutions: One solution is: 101, 102, 145, 224. Another solution is: 101, 145, 102, 224. Here is a pseudo-code description of a topological sorting algorithm:

Definition:

We say that course B is a successor of course A if course A is a prerequisite of course B. (For example, 102 is a successor of 101, since 101 is a prerequisite of 102.)

Algorithm:

TopologicalSort(courseList):
// The parameter courseList is a list of courses.
// Each course stores its prerequisites and successors.
1. Initialize a queue q to hold all the courses in courseList that have no prerequisites.
2. While the queue is not empty do:
   a. Remove a course c from the queue.
   b. Print out the name of course c.
   c. For each successor p of c do the following:
      i. Remove c from the prerequisites of p.
      ii. If p now has no prerequisites, then add p to q.
Implementation:

1. Make a data structure to represent computer science courses and their prerequisites: Define a method `initializeCourses()` that returns an `ArrayList<Course>` that holds all the courses offered by the Computer Science Department of Vassar College. (Check the Java 1.6 API documentation online to see how to use the `ArrayList` class.) Include only the courses: 101, 102, 145, 203, 224, 235, 240, 241, 245, 250, 324, 325, 331, 334, 365, 366, 375, 377, 378. Your `initializeCourses` method should repeatedly call the `addPrerequisite` method of the `Course` class to add appropriate prerequisites for the courses being defined. The `Course` class is already defined for you. The constructor for the `Course` class takes the name of a course (e.g., “CMPU101”) as its only argument. Look at the `Course` class definition to see how the `prerequisites` are added/removed to/from and stored in the `Course` object.

2. Define a method `markSuccessors(ArrayList<Course> courseList)` that records the successors of each course, assuming that all prerequisites have already been defined. It should operate as follows: For each course `c` in `courseList`, this method should take each prerequisite `p` of `c` and call `p.addSuccessor(c)` to add `c` to the successors of `p`.

3. Define a method `initializeQueue(ArrayList<Course> courseList)` to return a `Queue<Course>` holding all courses with no prerequisites. The queue should be implemented as a `LinkedList<Course>`. This method should initialize an empty queue; iterate over the members of `courseList`, identify the courses that have no prerequisites, add each of them to the queue; and return the queue. (You should use the `Queue` interface and the `LinkedList` class defined in the Java 1.6 API.)

4. Implement the `void topSortCourses(ArrayList<Course> courseList)` method according to the pseudocode definition shown above.