Package Installer Exercise

Description
You suddenly have a curious aspiration to create a package installer that can handle dependencies. You want to be able to give the installer a list of packages with dependencies, and have it install the packages in order such that an install won't fail due to a missing dependency.

This exercise is to write the code that will determine the order of install.

Requirements
1. Please complete the exercise in either C# or Javascript.
2. Please use Test Driven Development (TDD) and include your tests.
3. Please submit your code in a git repo (zipped and emailed, not on github) where you have committed throughout the process so that we can see your progress as you code the solution.
4. The program should accept an array of strings defining dependencies. Each string contains the name of a package followed by a colon and space, then any dependencies required by that package. For simplicity we'll assume a package can have at most one dependency.
5. The program should output a comma separated list of package names in the order of install, such that a package's dependency will always precede that package.
6. The program should reject as invalid a dependency specification that contains cycles.

For example, the input

```
KittenService: CamelCaser
CamelCaser:
```

represents two packages, "KittenService" and "CamelCaser", where "KittenService" depends on "CamelCaser". In this case the output should be

```
CamelCaser, KittenService
```

indicating that CamelCaser needs to be installed before KittenService.

Example of valid input
KittenService:
Leetmeme: Cyberportal
Cyberportal: Ice
CamelCaser: KittenService
Fraudstream: Leetmeme
Ice:

A valid output for the above would be:

KittenService, Ice, Cyberportal, Leetmeme, CamelCaser, Fraudstream

Example of input that should be rejected (contains cycles)
KittenService:
Leetmeme: Cyberportal
Cyberportal: Ice
CamelCaser: KittenService
Fraudstream:
Ice: Leetmeme

Review
Please submit your code in a git repo (zipped and emailed, not on github) where you have committed throughout the process so that we can see your progress as you code the solution. We will be running your code against our internal set of tests. We'll get back to you with next steps after we review your submission.