2. Your starting point is the same array-based implementation as in the previous assignment. You may make use of my sample solution code for that assignment if you wish.

3. Rewrite the interface and implementation of `ResearchPlan` so that

    - The three data members `maxRequirements`, `requirements` and `numberOfRequirements` are replaced by a vector of `Topic`, named `requirements`.

        > `requirements` and `researchTopic` must be the **only** data members in the `ResearchPlan` class.

        - In manipulating the `vector`, you should do so in typical `vector` style. Do not pre-allocate blocks of unused elements as space in which to add prerequisites. Instead, let the vector grow as needed, so that the `size()` of the vector is always the number of prerequisites in the cour

    - A constructor is added to allow ResearchPlan objects to be constructed from a course name and a pair of iterators denoting a range `Topic` values in some arbitrary container.

    - An iterator (and const iterator) are added to `ResearchPlan`. These should provide access to the prerequisite `Topics`, stored within th `ResearchPlan` objects.

        - The iterators replace the use of the index-based access previously provided by the `getPrereq(int)` function, which should be removed.

4. Replace as many loops as possible in the `ResearchPlan` function bodies by iterator-based loops (possibly including range-ba loops) or by appropriate use of `vector` functions.

5. Your code will be evaluated both on its ability to function correctly within the `permittedResearch` application and on its abi to pass the various unit tests provided. As before, alternate tests will check first for correct output/behavior of your code, an for correct output with no memory leaks.

6. As in the prior assignment, you will receive an automatic grade report shortly after each submission. That will not, however include points for successfully rewriting the loops with iterator-based replacements. Those will be assessed manually by the instructor after the due date has passed, and resubmissions for the purpose of improving that part of your score will not be permitted.

2 Problem Description

Many computer games and simulations, including the genre of *4X* (Explore, Expand, Exploit, & Exterminate) games, feature a lon list of topics that must be researched in order to take advantage of more and more powerful in-game technology. These are often referred to as the game's *research tree* or *technology tree*. (Although, as we will see later in this course, the underlying structure of this data is not a "tree" but an "acyclic graph".)

Within the game, there is generally extensive documentation of the research tree, often described in a faux *Encyclopedia*.

A good example of such a research tree is the one for FreeCiv. Associated with each topic under "Tech Requirements" is a list of p topics that must be researched before the new topic can be studied, and a cost (measured in "bulbs") to research the new topic.

- For example, the topic "Advanced Flight" costs 44370 bulbs and can only be researched if "Radio" and "Machine Tools" ar already known.
- On the other hand, the topic "Alphabet" costs only 28 bulbs, and has no prior research requirements.

This program should read in an encyclopedia describing a series of *research plans*. A research plan consists of a "target" research topic and a list of prior topics that must be researched before research can begin on the target.

Note that the research plans are compressed in that, if topic A requires prior research of topics B and C, but topic B also requires C then only topic B is listed as a requirement of A. The fact that C is also a prior requirement of A can be inferred from the fact that C required for B. So while it would not, technically, be incorrect to list the plans:

```
A: B, C
B: C
```

we can and, usually, will go for the simpler set:

```
A: B
B: C
```

Next the program will read the list of research topic names already researched by the game player.

ıen the program will print the list of new topics that the play may select among as new topics of research. It will also note which of ɔse available topics is cheapest.

.1 Input

ıe input occurs in two distinct sections. First is the encyclopedia, then the list of already known topics. The two sections are parated by a single line containing only "—" (three hyphens). The second section can be terminated by another line containing ree hyphens or by the end of the input stream.

1.1 The Encyclopedia

ıe encyclopedia is described as a series of 1 to 1000 research plans, one plan per line.

plan is described by a line of the form

```
targetTopic: prior topic 1, prior topic 2, ... , cost
```

ıere may be 0 to 10 prior topics. The target topic is separated form the prior topics by a colon (:), and the prior topics are separated ɔm one another and from the cost by commas. Blanks may be inserted after the punctuation and will be ignored.

ıpic names may be any mixture of alphanumeric characters and blanks, but must have at least one non-blank character, must not gin with a blank, and must not include the characters ',' or ':'.

ery prior topic will be listed in some other line as a target topic for another research plan. No topic will be the target for more than ıe plan. The research plans will not contain any "cycles" whereby a topic winds up being required before that same topic can be ɜearched.

1.2 The Known Topics

ter the end of the encyclopedia, the already known topics are listed, one per line. There will be no leading blanks. Every known ɔic will occur only once, and every known topic will be the target of some research plan.

ıe number of know topics will be in the range `0...N`, where `N` is the number of research plans.

2.2 Output

Output consists of

1. A line announcing the number of topics on which research could begin.
2. Followed by a list of those topics, in alphabetic order, together with their research costs.
3. Followed by a suggestion as to the cheapest topic on which research can begin. If there is a tie for cheapest researchable topic, the program may select any one from among the tied list.

A more detailed description of the output format is not provided because you will not be responsible for the output format, only for parts of the calculations leading up to that output.

If you add debugging output to the code, you must remove or comment it out before submitting. Extra output will be flagged as incorrect.

2.3 Examples

For example, given the input

```
Alphabet: 28
Bronze Working: 28
Ceremonial Burial: 28
Code of Laws: Alphabet, 79
Currency: Bronze Working, 79
Horseback Riding: 28
Iron Working: Bronze Working, Warrior Code, 136
Map Making: Alphabet, 79
Masonry: 28
Mathematics: Alphabet, Masonry, 136
Mysticism: Ceremonial Burial, 79
Polytheism: Horseback Riding, Ceremonial Burial, 136
Pottery: 28
The Wheel: Horseback Riding, 79
Warrior Code: 28
Writing: Alphabet, 79
```

(available as test0.in in the assignment files, the output should be

```
There are 8 topics available for research.
                Alphabet  28
                Currency  79
            Iron Working  136
                Masonry  28
               Mysticism  79
              Polytheism  136
                 Pottery  28
              The Wheel  79
The cheapest to research is Alphabet: 28
```

Given the input:

```
Alphabet: 28
Bronze Working: 28
Ceremonial Burial: 28
Code of Laws: Alphabet, 79
Currency: Bronze Working, 79
Horseback Riding: 28
Iron Working: Bronze Working, Warrior Code, 136
Map Making: Alphabet, 79
Masonry: 28
Mathematics: Alphabet, Masonry, 136
```

```
Writing: Alphabet, 79
---
Bronze Working
Ceremonial Burial
Horseback Riding
Warrior Code
Alphabet
Masonry
Pottery
---
```

(available as test1.in in the assignment files, the output should be

```
There are 9 topics available for research.
            Code of Laws  79
                Currency  79
            Iron Working  136
              Map Making  79
             Mathematics  136
               Mysticism  79
              Polytheism  136
               The Wheel  79
                 Writing  79
The cheapest to research is Code of Laws: 79
```

A larger example (test2.in) is also found in the assignment files. You should run my solution if you want to see the expected output.

2.4 Running the Program

2.4.1 System Tests

The `permittedResearch` program takes its input from a file named in the command line and produces its output on standard out (`cout`) For example, to run the program on the provided sample `test0.in` file, the command would be

```
./permittedResearch test0.in
```

```
test0.in
```

Windows, use ".\" instead of "./". Of course, if you have your files in different directories
to adjust your paths in the above command accordingly.

e when running within Eclipse, refer to the FAQ list.

omit the file name parameter, in which case the program will read from standard in (`cin`).

n a separate executable, named `unittests`. It can be run without parameters to run all tests
ses. (Note that some bugs in the `ResearchPlan` code could cause `Encyclopedia` tests to fail,
n the `ResearchPlans`.

one or more test names as parameters to run those tests only. For example, if you are faili
t test, run

```
hPlanConstruct
```

e test. This can be very useful when debugging your code, because the unit tests are simpl
le a great deal more directly than the full application.