Class Relations

Purpose: Trees

Due: Feb 20th, March 6th

In object oriented programming classes can be related through inheritance. One class (the subclass class) can be derived from another class (the base class). In java we would say the subclass *extends* the base class. The subclass has all the members (functions and data) from the base class. The subclass usually has more members and is a specialization of the base class. For example, a `Date` class might be derived from `Time` class. Then if the `Time` class has members for `hour, minute and second` , then the derived `Date` class could have members for `day, month and year` as well as `hour, minute and second`. Further, `Appointment` might be derived from `Date` with additional fields for `location and name`. This example, (pictured below), an instance of an appointment has fields for hour, minute and second.

```
class Time       class Date : public Time    class Appointment  : public Date
{                 {                         {
public:           public:                    public:
  int hour;         int day;                  string location;
  int minute;       int month;                string name;
  int second;       int year;
    :                   :                                      :
 }                 }                         }
```

Figure 1: Inheritance in C++ classes.

Note that this relationship is transitive. In other words if A extends B and B extends C then A extends C. Also note that a class can inherit from multiple classes.

In this problem you will be given a set of extension relationships and a set of queries of the form A extends B or A isExtendedBy B. You must determine if each query is true or false.

Part 1 Due Feb 20

Input:

Input starts with one integers n , $(1 \leq n)$, where n specifies the number of given *extends* The next n lines each contain one given relationship in the form c_1 c_2 where c_1 and c_2 are single-word class names.

Output

Once the input is read, each of the base classes and the classes that extend it. Each extension should be indented 5 spaces. Between each base class tree print 10 dashes.

Example Input

```
10
Date Time
Appointment Date
Calendar Appointment
Circle Point
Triangle Shape
Rectangle Shape
Square Rectangle
Cone Circle
Cylinder Circle
Circle Shape
```

Corresponding Example Output

```
Time
     Date
          Appointment
               Calendar
----------
Point
     Circle
          Cone
          Cylinder
----------
Shape
     Triangle
     Rectangle
          Square
     Circle
          Cone
          Cylinder
----------
```

Starter code

```
/*
Class relations
by <Your name goes here>
*/
class Node {
public:
  Node(string label) :name(label) {}

  Node * find(string toFind) { // returns a pointer to the node with name == toFind
  }                            // returns nullptr if not found
  void print() {               // prints the tree rooted at this
    print(0);
  }
  void addChild(Node *childPtr) {  // updates children/extends vectors
  }

  string getName() {
    return name;
  }
private:
  string name;
  vector< Node *> children;
  vector< Node *> extends;
};

class Program {
public:
  Node * find(string toFind) { // returns a pointer to the node in the forest with
    name==toFind
  }                            // otherwise returns nullPtr

  void print() {               // prints the forest
  }
  void add(string name1, string name2) {
                              // adds the relation name1, name2 to the forest
  }//add                        // 4 cases: both not in the forest , 1 in the forest ,
    both in forest
private:
  vector<Node *> classes;
};

int main() {
  Program program;
  string name1, name2;
  int numPairs;

  cin >> numPairs;
  for (int i = 0; i < numPairs; i++) {
    cin >> name1 >> name2;
    program.add(name1, name2);
  }
  program.print();
  return 0;
}
```

```
class Node:
  def __init__(self , className ): # constructor

    def find(toFind):# returns a pointer to the node with name == toFind
                            #returns None if not found

    def print( ): #prints the tree rooted at self

   def addChild(childPtr):#updates children/extends vectors
```

```python

class Program:
  def __init__(self ): # constructor

    def find(toFind):# returns a pointer to the node with name == toFind
                            #returns None if not found

    def print():
```

Part 2

Input:

Input starts with an integer n ($1 \le n$), where n specifies the number of given *extends* relationships. The next n lines each contain one given relationship in the form c_1 c_2 where c_1 and c_2 are single-word class names. The next line is an integer m ($0 \le m$) where m specifies the number of queries. The following m lines contain 1 querry per line, of the form c_1 r c_2, where r is either "extends" or "isExtendedBy". All class names in the last m lines will appear at least once in the initial n lines. All extends and isExtendedBy relationships between the given classes can be deduced from the n given relationships. Extends relationships can not be circular (apart from the trivial identity "x extends x").

Output

For each query, display the query number (starting at one) and whether the query is true or false. Follow this by a list of all the classes (in alphabetical order) 1 per line, where each class is followed by the names of the classes that it extends.

Sample Input

```
10
Date Time
Appointment Date
Calendar Appointment
Circle Point
Triangle Shape
Rectangle Shape
Square Rectangle
Cone Circle
Cylinder Circle
Circle Shape
8
Time extends Date
Calendar extends Time
Cylinder extends Circle
Cylinder extends Shape
Time isExtendedBy Date
Calendar isExtendedBy Time
Circle isExtendedBy Cone
```

```
Time isExtendedBy Shape
```

Sample Output

```
1 false
2 true
3 true
4 true
5 true
6 false
7 true
8 false
Appointment Date Time
Calendar Appointment Date Time
Circle Point Shape
Cone Circle Point Shape
Cylinder Circle Point Shape
Date Time
Point
Rectangle Shape
Shape
Square Rectangle Shape
Time
Triangle Shape
```

How the program will be graded

Memo

What	pts	Due
Name	1	
Time Analysis O() of every function[1,2] (in terms of the words input	5	March 6
Space Analysis O() of every function[1,2]	5	March 6
A class diagram	10	March 6

Source Code Document

[1]The main() is a function.
[2]All analysis should be worst case based on the number of input words.
[3]A test plan is a table with 4 columns and 1 row per test. The columns are named Reason for the test, actual input data, expected output data, and actual output. You do NOT have to have a working program to write a test plan. Each reason should be unique.
[4]A non trivial test contains only legal data (data the conforms to the input specification) with graphs contain at least 1 vertex .

What	**pts**	**Due**
Name	1	Feb 20
Description[5]	4	Feb 20
Style	8	March 6
pre/post conditions	7	March 6
Functionality	50	
`Node.find(string)`	4	Feb 20
`Node.print()`	4	Feb 20
`Node.addChild(childPtr)`	1	Feb 20
`Program.find(string)`	4	Feb 20
`Program.print()`	1	Feb 20
`Program.add(derivedName, baseName)`	8	Feb 20
Processes Queries	28	March 6

[5]The description should be written to some one who knows NOTHING about the program. It should discuss what the program does (in your own words). After reading the description the user should be able to create legal input and predict the output.