

# CSC 1800

## Organization of Programming Languages

### Fall 2018

#### Programming Assignment #2

**Out:** 26 September 2018

**Due:** 11:59:59 pm, Sunday, November 4, 2018.

#### Introduction

This project deals with *family trees*. A family tree is a set of individuals who are “connected” to each other by relationships. You will write a Java program to solve the problem of determining how people are related to each other. You will write a Python program to solve the problem of determining how two people are related to each other. We will be using Python 3 in this project, NOT PYTHON 2. You can get the Python interpreter at <http://www.python.org>.

#### Problem Background

For this problem, you must design and write a Java program that will read an input stream that defines a family tree, and then answer questions about different specializations of “*related*” between people in the tree. There are two kinds of questions you will handle:

A) “*IS-A*” Questions: These are “Yes/No” questions about whether two people <A> and <B> share the following relationships:

1. Is <A> a child of <B> (order of names matters!)
2. Is <A> a spouse of <B>
3. Is <A> a sibling of <B>
4. Is <A> an ancestor of <B>
5. Is <A> a cousin of <B>
6. Is <A> unrelated to <B>

B) “*WHO-IS-A*” Questions: “Who are all the people who are A’s <relation>?”  
<Relation> can be any of the relationships listed for “IS-A” Questions.

Another way of looking at a family tree is to view it as a directed graph. The nodes represent individuals, connected by edges (arrows) that represent the “offspring” relationship. An edge from one node to another means that the latter node is an offspring of the former node. In general, we can say one node (individual A) is a *direct ancestor* of another node (individual B) if and only if (iff) there is a backward sequence of one or more edges from node B back to node A.

#### *Defining the general concepts of “related” and “unrelated:”*

Nodes in a family tree have at most two edges in (one for each biological parent), and an arbitrary number of edges out (one for each offspring). Two people are *related* if they share a common ancestor OR if one is a direct ancestor of the other. Thus, if two people have no common ancestor AND neither is the direct ancestor of the other, then they are *unrelated*.

#### *Defining the cousin relationship*

Another way of looking at the *cousin* relationship is that it is the same as the first part of the *related* definition: two people are cousins iff they *share* a common ancestor, (but neither is a direct ancestor of the other). Thus, a parent and a child are not considered cousins, nor would grandparents and their grandchildren be considered cousins.

### *Re-Marriages*

A twist in this problem is that you'll have to be careful of second, third, fourth, etc marriages – do not assume that a person can have at most one spouse. In the case of second marriages, technically the children from each spouse's earlier marriage are NOT siblings, but new children produced after the marriage are (half-)siblings of the earlier marriages' offspring. **In our project, however, we'll have to recognize that there can be remarriages, BUT two people with only a single parent in common are still siblings.**

Also, in this project, queries about spouses are not time-dependent. That is, once two people are married, they are still considered spouses by this program even if they get remarried. So, if you are asked WHO-ISA spouse of person X, you'll need to list all people who have ever been spouses of X.

### **Problem I/O Specification**

The family tree will be supplied to your program via **standard in**. This means you should be sure to open a `BufferedReader` or `Scanner` on the `System.in` stream. You should also be printing all output to `System.out`.

***NOTE: You must NOT write the program so that it always looks at some hard-coded data file name or so that you have to supply the name as a command line argument.***

You can assume that there will be no formatting errors to check for. Each line in the family tree will have the form

E <name1> <name2>      or      E <name1> <name2> <name3>

which has the meaning “<name1> and <name2> are married” or “the married parents <name1> and <name2> produced a child <name3>.” Note that marriage events and birth events are listed chronologically, and that not all marriages produce children. Names have no spaces in them, have no hyphens, and are under 40 characters long.

The family tree file will also have query lines (possibly interspersed with event lines). These will have the form

#### QUERY

X <name1> <relation> <name2>

W <relation> <name1>

<relation> can be **child, spouse, sibling, ancestor, cousin, or unrelated**.

#### Meaning

Is <name1> the <relation> of <name2>?

List everyone who is the <relation> of <name1>.

Whenever the program encounters a query, it must answer according to the knowledge so far collected as events in the input file.

No output is required when the program encounters an event line. When a query line is encountered, the program must print out a blank line, then print the query, then on the next line print out the response. The response to “X” queries must be either “Yes” or “No.” The response to “W” queries must be all the names, one per line, that correctly answer the question. The names must be sorted in alphabetical order.

## Example I/O

### File Content:

E John Mary Bill  
E John Mary Pete  
E John Mary Fred  
E John Jean Rebecca  
E Rebecca Bill Andrew  
E Pete Carol Jim  
W child John  
X Bill sibling Pete  
X Bill sibling Fred  
X John unrelated Mary  
W ancestor Andrew  
X Bill cousin Mary

### Output:

W child John  
Bill  
Fred  
Pete  
Rebecca

X Bill sibling Pete  
Yes

X Bill sibling Fred  
Yes

X John unrelated Mary  
Yes

W ancestor Andrew  
Bill  
Jean  
John  
Mary  
Rebecca

X Bill cousin Mary  
No

## Project Hints

You'll first need to decide how to represent people – what kind of data structure will you use? PLEASE DO NOT USE OBJECT ORIENTED TECHNIQUES FOR THIS PROJECT. What components should “people objects” have? You'll then have to decide how to organize all of the “people objects” into a graph. Remember that lists or arrays in Python can themselves hold things like linked lists and arrays. You also must be careful to set procedures up so that you can easily add births/marriages/parents/children to the graph as you read in new events. I'd recommend a hash table or an array of people structures. You'll also need to write functions or procedures for answering each of the queries. Most of these will be some kind of graph-traversal algorithm (breadth-first or depth-first), although some could be simple checks of the contents of a single component on a person. Note that you'll need to keep around a linked list of nodes seen so far during your traversals when you're looking for common ancestors! Finally, you might find that you don't need a procedure for each query, but rather a smaller number of very generic procedures might be able to solve all of the queries.

You can choose how to schedule the design of your project as you see fit.

To help you in your planning, I **suggest** that you use the **first week** of the assignment to

- a) identify the primary methods and objects you'll need to represent people and retrieve them from a hash table (or array) and create and update them.
- b) define methods to verify if two people are related according to the relationships defined above. There should be a method for each of the queries

By the end of the second week at the latest, you should have the first category of question fully tested and implemented.

By the end of the third week, you should have the second category of question implemented. This schedule makes sense because the second category really just uses the methods for your first category to test to see if a person is of the specified relationship.

You should be testing your code with a set of test files that you make up. Each time you add a new feature, please re-run the tests to see that you haven't caused any problems with your new code that causes old tests to fail. This is called regression testing.