Programming Assignment: Families

1. Background

Genealogical databases can be represented by directed bipartite graphs. Families and persons are the two kinds of vertex; each is connected only to the other. The edges from a family vertex to person vertices are labelled "husband", "wife", and "child"; there can be multiple "child" edges. The edges from a person vertex to family vertices are called "parentage" and "marriage".

Genealogical data are always incomplete. There are individuals whose parents are unknown; their vertices have no "parentage" edge. There are families for which the husband or wife is unknown; their vertices have no "husband" or "wife" edge. There are families with no children yet, or whose children are unknown; these families have no "child" edge, or only an incomplete set of "child" edges.

2. The assignment

Write a program called "genealogy" that takes the following commands:

```
Family n Husband n Wife n Child n Child n ...
Relate n n
```

Here, `n` stands for an integer. Each command is on its own line.

The `Family` command introduces a family vertex and connects it to person vertices by "husband", "wife", and "child" edges. It also introduces person vertices as necessary and connects them to the new family vertex via "parentage" or "marriage" edges. The `Family` command mentions as many children as necessary; a family may have no children. For instance, `Family 13 Husband 0 Wife 3 Child 7 Child 9` means "Family 13 has unknown husband, wife 3, and children 7 and 9." Persons and families are numbered independently; family 3 is unrelated to person 3. When it receives a `Family` command, the program should output a line like "Family 13 has husband 0, wife 3, and children 7 9."

The `Relate` command asks for the shortest path between two persons. For instance, `Relate 5 8` asks how persons 5 and 8 are related to each other. If they are not in the same connected component of the graph, the program should print a line like "Persons 5 and 8 are not related." The program must use Union-Find to check connectedness before using breadth-first search to find a path. If they are in the same connected component, the program should print a line showing the shortest path connecting them, in the form: "Relation: person 5 -> family 2 -> person 18 -> family 6 -> person 8". If there are several shortest paths, it only prints one of them.

The program must disallow these situations, printing an error message and ignoring commands that would cause them: (1) A person having multiple marriages (that is, being husband or wife more than once). (2) A person having multiple parentages (that is, being a child of more than one marriage). (3) Creating a family with a number that already refers to a family.

3. Assumptions

You may make the following assumptions.

1. All commands are properly formatted. You might still want to check input format to verify your own data.
2. The `Family` command never mentions a child numbered 0.
3. No family has more than 10 children.
4. There are no more than 99 persons and no more than 99 families. Person numbers and family numbers are in the range 1 .. 99. Each number can be used twice, once for a person and once for a family; there is no implied

connection between a person and a family that happen to have the same number.

4. Useful tools

As always, you have access to some useful tools. First, there is a sample `Makefile` at http://www.cs.uky.edu/~raphael/courses/CS315/prog5/Makefile. It has a *run* target that compiles your program (either `genealogy.c` or `genealogy.cpp`) and runs it.

You can also get a working program that satisfies the specifications at http://www.cs.uky.edu/~raphael/courses/CS315/prog5/workingGenealogy. The `Makefile` mentioned above automatically gets a copy of this file for you if you make *runWorking*.

5. What to hand in

Your submission should include your program, all documentation, a Makefile, your program's output on the data in http://www.cs.uky.edu/~raphael/courses/CS315/prog5/data.txt, your own test data, and your program's output on that test data.

6. Extra-credit ideas

1. Allow persons to have multiple marriages.
2. Allow same-sex marriages.
3. Notice if a `Family` command introduces a valid new connection (like a cousin marriage) between persons who are already connected.
4. In addition to the required output, generate a nicer output for the `Relate` command, such as "Person 5 is a husband in family 2, which has a child person 8, who is the wife in family 6, which has a husband person 4" or, even better, "5's child 8 is wife of 4".
5. Allow the `Person` and `Family` commands to take extra information, such as (for persons) name, birth date, residence, and (for families) marriage date. Implement new commands `Describe Person n` and `Describe Marriage n` that show this information.
6. Show *all* paths connecting two persons, shortest first.
7. Implement the `Descendants n` command, which displays all the descendants of person `n`.
8. Implement the `Ancestors n` command, which displays all the ancestors of person `n`.