CSC461 Programming Languages – Fall 2017
Programming Assignment #4: Genealogy (Lisp)

Problem

Write a Lisp program that reads in a file of family relationships, builds a database, and then allows queries to that database. Queries include: *parents, mothers, fathers, children, sons, daughters, siblings, sisters, brothers, grandparents, grandfathers, grandmothers, grandchildren, grandsons, granddaughters, uncles, aunts, nieces, nephews, cousins, male-cousins, female-cousins, ancestors, male-ancestors, female-ancestors, descendants, male-descendants,* and *female-descendants*.

Implementation

Your Lisp program should read in family relationships from a file given on the command line. Each file entry contains a list of information about a person: name, gender, parents, and children. For example, if Bob (male) has children named Matt and Becky, and parents named Sam and Doris, then the corresponding file entry would be:

```
(bob male (sam doris) (matt becky))
```

Use Lisp *structs* to create and access the genealogy database. The only fields allowed are *name, gender, parents,* and *children*. The *name* field stores the person's name; the *gender* field is either *male* or *female*. The *parents* and *children* fields are lists consisting of zero or more names.

```
(defstruct person name gender parents children)
```

Your program should include a function which reads information from the family tree file, builds the database, and stores it in a global variable called **database**. This function should be called automatically when the program is run from the command line. The database may then be queried by typing requests such as

```
> (children 'bob)
(matt becky)                    ; Lisp response
```

A database query returns a list of zero or more names that correspond to the specified family relationship.

Program usage: `clisp –repl genealogy.lsp family.dat`
where *family.dat* is the name of the database file.

Database Details

- The database *must* be referenced by a global variable named **database**.
- The database may not be complete; i.e., some relationships may not be filled in.
- You may assume that the database is consistent. (Duplicate names are allowed in the database, but may give inconsistent results.)
- A person may have more than one set of parents.
- Be careful not to return duplicate or inappropriate names in response to a query. For example, a person is not considered to be his/her own sibling.

- Extract as much information as you can from the database, without going to unreasonable extremes. For example, uncles and aunts should include parents of cousins as well as siblings of parents. However, you do not need to go up and down the family tree multiple times. For example, siblings should include children of parents, but you do not need to find the children of parents of children of parents, even though that might add additional family members.
- To eliminate duplicated code for male and female family relationships, define the functions *malefilter* and *femalefilter*.
- Make sure you spell all the function query names correctly!

Program submission notes

- Complete your program by the due date (Friday December 1) in order to receive credit for this assignment. Late programs will not be accepted for partial credit unless prior arrangements have been made with the instructor.
- To receive full credit, your code must be readable, modular, nicely formatted, and adequately documented, as well as complete and correct. It must build and run successfully using the current CLisp interpreter (version 2.49, available on the course website) under Windows and Linux. If your program does not run correctly, indicate why. This will make it easier to give you partial credit.
- Work in teams of two students on this assignment. Your program will be pulled from your GitLab repository on the due date at midnight. Name the project *CSC461_F17_PA4*. Be sure to add your instructor as a developer on the project. This will enable me to monitor your progress and pull your project for grading.

  Partners for PA#4
  Ian Beamer + Ryan Hinrichs
  Christopher Blumer + Jeremy Goens
  Allison Bodvig + Chad Heath
  Joey Brown + Nathan Ducasse
  Kathleen Brown + Kyle MacMillan
  Noah Brubaker + Collin Chick
  Aaron Campbell + Lucas Carpenter
  Jake Davidson + Logan Lembke
  Darla Drenckhahn + Matthew Schallenkamp
  Benjamin Garcia + Naomi Green
  Aaron Gibbs + Katherine MacMillan
  Lawrence Hoffman + Ryan McCaskell
  Tanner Holthus + Andrew Housh
  Cameron Javaheri + Mathias Wingert
  Kyle Lorenz + Sierra Wahlin-Rhoades
  Soham Naik + Isaac Rath
  Garret Odegaard + Ryley Sutton
  Michael Pfeifer + Hannah Wegehaupt

When you finish the assignment, email me a brief description of team member contributions (both you and your partner) using the form *Teamwork Evaluation.docx*. Your comments will be kept private, but I may choose to grade team members individually.