CSCI 4220 Programming Languages
Programming Assignment Three

This program will be a functional program written in LISP. The program will allow the user to ask questions about family relationships, which the program will then answer. First let me describe the data. Here is a very simple example:

```
((Al Betty (Chuck)))
```

Al is married to Betty and they have one child named Chuck. The data structure is recursive, of course:

```
((Dora Ed (Frank Gloria (Hannah)
                        (Ingrid James)
                        (Karen Lou (Matt) (Nathan)))))
```

Dora is married to Ed. They have only one child named Frank. Frank married Gloria and they have three kids, namely Hannah, Ingrid, and Karen. Ingrid married James but they have no children. Karen married Lou and they have two kids, Matt and Nathan, neither of which is married. Dora is Hannah's grandmother and Ed is Matt's great-grandfather.

So in general, if I have a list with one element, it is one person. If I have a list with two atoms at the front, the first is the direct relation (child) and the second is the spouse of that person (son-in-law or daughter-in-law). If there is a third element, fourth element, etc. these must all be lists which represent *their* children, who may or may not be married, etc. The data structure does not allow for divorces, so there's no step-fathers, etc. And all names in the family will be unique.

The assignment!

Suppose I have an involved data structure in the above format. Your program should support the following functions:

`(children`	This function takes one parameter and returns a list of the children of that person. For example, `(children 'Frank)` returns `(Hannah Ingrid Karen)`
`(g-children`	This function does the same thing, except it returns a list of the grand-children of the parameter.
`(g-g-children`	Same thing again, but great-grand-children.
`(parents`	This returns a list of the two parents of the individual. For example: `(parents 'Matt)` returns `(Karen Lou)`. The first of these is the offspring of the grand-parent.
`(g-parents`	Same thing, but returns the grand-parents.
`(g-g-parents`	Same thing, but returns the great-grand-parents.
`(sibling`	This function returns a list of all of the siblings of someone. For example, `(sibling 'Matt)` returns `(Nathan)`. Note that the list does not include Matt.

`(cousin`	This function returns a list of all of the cousins of someone. In the above data there are no cousins, but a cousin is someone who has the same grandparents.

I will give you a larger data set to test with. **Please name your variable “family”, as in:**

```
; Here is the family we want to use.
(setq family '((Dora Ed
                     (Frank Gloria (Hannah)
                                   (Ingrid James)
                                   (Karen Lou (Matt) (Nathan)))
                     (Olive)
                     (Patrick Quady (Renee)))))
```

(Oh, wait! I can now state that `(cousin ‘Renee)` should give `(Hannah Ingrid Karen)`. At least I think so.)

This way I can do `(load ‘yourprog)` and then `(load ‘myfamily)` to overwrite the value you have set. Remember that LISP will let me do this – I can “`setq`” something *after* you have.

This program is due at midnight on April 23rd. Please note that this is the same date as program 1.

What to hand in when: Well, the program is due at midnight on April 23rd. That is the last class before dead week. The programs are automatically collected at that time and it is important that they are in the correct place. So! Please put all of your source code in one directory named “PL_Spring09_prog3”, and within that directory put all of the source code. For example, in my case I would have a directory named “/home/wmahoney/ PL_Spring09_prog3” and in that directory would be my source code named (for example) “prog3.lsp”. All of the contents of this directory will be automatically collected at midnight on the due date. I always get asked whether this means the midnight in the morning of the date or the one after – if you are sitting there *on April 23rd* and you’re watching David Letterman, your time is almost up.