

UNIVERSITY OF CALIFORNIA  
Department of Electrical Engineering  
and Computer Sciences  
Computer Science Division

Prof. R. Fateman

Fall, 2005

CS 164 Assignment 1: Learn Common Lisp Now!

**Due:** Thursday, Sept 8, 2005, 11:59PM

These assignments can be done in many different ways on different computers using different editors and lisp systems. I highly recommend that you use a version of emacs and common lisp that work together so you can edit in a buffer and execute code in another buffer. Gnu-emacs and Allegro Common Lisp are what I use. You can easily reduce your productivity to 50% or even 10% by unwise choices of editors and environments. Gnu-emacs and Lisp are available on Windows or UNIX systems, and you can run problems at home if you wish. We suggest you frequently copy any files you have at home on a UC system where it is backed up. We do not require that you use any “version control” programs.

1. In this problem you must write a simple function called `pairup` taking two lists of the same size, and producing a new list with the two lists intermingled in the following way: If `r = (a b c)` and `s = (1 2 3)` then `(pairup r s)` results in the list `((a . 1) (b . 2) (c . 3))`. Such a list is commonly called an association list or a-list for short.

2. Write a (recursive) version of the LISP function `assoc`. Call it `myassoc`. `Assoc` “looks up” a value in an a-list, and returns the first sub-list it finds whose first element matches the desired value. For example, `(myassoc 'y '((x . a) (y . b) (z . c)))` returns `(y . b)`

3. Using your definition or the built-in one for `assoc`, write a program `change-binding` that alters such an a-list. If you were using Scheme, you might use `set-cdr!`, but in Lisp you would use the somewhat magical form `(setf (cdr x) newval)`. That is, if you do

```
(setf r '((a . 1) (b . 2) (c . 3)))  
(change-binding 'b 5 r)
```

The new value for `r` is `((a . 1) (b . 5) (c . 3))`. The value returned from `change-binding` in this case should be 5. Learn enough about `format` so you can print out an error message in case the look-up fails.

4. Use hash-tables (a built-in facility in COMMON LISP) to perform the same kind of operations: Define a function `hash-pairs` that takes two lists: of keys and values, and inserts them into a hash-table, returning the table; also define a `hash-assoc` function (compare it to `gethash`), and a `hash-change-binding`.

5. Professor North D. Coder is an undergraduate advisor for 40 students. Since he sees them about twice a year, he doesn't recognize them all. He asks an advisee who walks into his office during office hours: "What is your name?" and her response is "Jane". Since the file box from the registrar is ordered by LAST name, he needs to know Jane's last name. Prof. Coder is too embarrassed to ask Jane this vital information, and so he turns to his workstation and types in `(whois '(jane ?))`. He expects to get one of three kinds of answers.

1. `jane bancroft last visited you on 10-1-01.`
2. `You have no advisee with name (jane ?) on file.`
3. `You have 2 advisees with name (jane ?): ((jane bancroft 9-10-01) (jane oxford 10-1-01)).`

If the third possibility occurs, Prof. Coder can say to Jane, "Forgive me, I've forgotten your last name." If Jane says, "My last name is Oxford," Prof. Coder then knows when Jane Oxford last visited, and where he can find her file folder in his advising box.

The data base that Prof. Coder maintains looks like an association list:

`((jane bancroft 9-10-01) (jane oxford 10-1-01) (herman durant 11-1-01) ...).`

Write the `whois` program. You will need to write a simple matching (or "unification") program so that inquiry `(jane ?)` and data `(jane oxford)` match.

Sometimes a student gives only his last name. Set up the program so that the professor can also type `(whois '(? Hearst))`.

Your program should do something sensible for `(whois '(? ?))`.

Do you make the assumption that Prof. Coder does not have two students with the same first and last name? If so, where?

6. Read the file below, and change the program `showfile` so it provides page and line numbers. Assume that a page holds 66 lines. You can "eject a page" by `(format t "~c" #\page).`

7. Change `showfile` again so that another file can be used for output (other than standard output). Extra: make it work so that if the output file is omitted, it sends to `stdout`. If you can't figure out files and optional arguments in a few minutes of looking at the Graham book or online documentation, ask for help. Actually, this advice goes for almost every assignment.

inserted here ... the file `reading.cl` from

<http://inst.EECS.berkeley.edu/~cs164/software/source/reading.cl>