CS 63 AI – Introduction to Lisp Lab

This lab gives a basic introduction to Lisp, and will provide the foundation for you to interact with the Lisp interpreter.

In this class, we will be using the `clisp` (`/usr/bin/clisp`) implementation of Common Lisp that is installed on the CS machines. The course webpage gives instructions for how to install clisp on your own machines.

1.) Start `clisp`. At this point, you are immediately ready to interact with the interpreter.

Note: Whenever you make an error, you are immediately launched into the debugger. It is very easy to get lost in nested levels of debugging and correction. To back out a level, type `:a`. To get out of lisp, type `(exit)`.

2.) Lisp can automatically record everything you do in your session using the `dribble` command. Type `(dribble "lab1dribble.lisp")` to automatically record everything in the specified file. This will come in handy to remind you of your work later and also to demonstrate the output of a lisp program.

3.) Try executing a few lisp commands to see what they do. These will be logged to your dribble file.

a.) `(expt 10 3)`
b.) `(* 2 (- 10 7))` ← lisp uses prefix notation
c.) `(car '(a b c))` or, equivalently, `(first '(a b c))` ← (a b c) is a list
d.) `(cdr '(a b c))` or, equivalently, `(rest '(a b c))`
e.) `27`
f.) `(setq 'a '())`
g.) `a`
h.) `(setq a '(b c))`
i.) `(push 'd a)` ← Stacks are easy in lisp!
j.) `(pop a)`
k.) `a`
l.) `(setq a '(- (+ 2 3) 1))` ← You can write and execute lisp code
m.) `(eval a)` ← easily within lisp
n.) `(listp a)`
o.) `(car (cdr a))`

4.) Exit out of lisp and use `less` to look at your dribble file. Everything should be there. Don't worry about mistakes you may have made while typing. There is no need to clean up the file.

5.) Make a new file called "average.lisp" using your favorite text editor. Let's write a lisp function that computes the average of a list of numbers. We won't worry about error checking inputs for right now. Here's a simple version:

```
(defun average (numList)
   (/ (apply #'+ numList) (length numList)))
```

Type this in the file and save it. Go back into lisp and load the file for execution using `(load "average.lisp")`. Now, try out the function.

a.) `(average '(1 2 3 4 6))`
b.) `(average '(1 2 3 4 6.5))`

The function `apply` allows us to apply an arbitrary function to the list. All we need is the function's name and we can use it. The following example takes the average of each sublist:

```
(mapcar #'average '((1 2) (3 4) (5 6)))
```

Note how the function itself is actually an argument to another function.

6.) Lastly, we'll write a function to compute the *n*th Fibonacci number together. To do this, we'll need two more constructs:

`(eql n 0)` ➔ a predicate that tests whether *n* is 0

```
(cond (predicateA X)
      (predicateB Y)
      (predicateC Z)
      ...)
```

`cond` is an if-test structure. It first tests if *predicateA* is true. If it is, then *X* results. If not, it tests if *predicateB* is true. If so, then *Y*. And so on.

Be sure to save both your dribble and "average.lisp" files. Next week, we'll use the `handin63` utility to submit them for credit for this lab and also to test that everything is working in preparation for homework 1.

Have a good weekend!