CSCE 476/876 Lisp Tutorial 1 Spring 2008

Jamie Schirf

This tutorial will take you through some of the basics of the Lisp language. Everything covered herein can be found within the first 3 chapters of the Winston/Horn LISP book. You should run these commands interactively through emacs.

1) Basic Numeric Functions

Enter
(+ 2 3)

It gives the sum of 2 and 3, 5. In LISP, + is a function, that adds all its arguments and returns their sum. Like all functions in LISP, it is the first element in a list, and the elements that follow it are its arguments. Of course, there are plenty other arithmetic functions in LISP. There's also -, *, /, along with others.

Try to predict the results of these commands and then run them to see if you're correct:

1.) **(+ 1 2 3 4)**
2.) **(* 4 (+ 2 2))**
3.) **(- 10 5)**
4.) **(- 10 5 5 5)**
5.) **(/ 4 3)**
6.) **(/ 4.0 3.0)**

Note what was done with entry 2. We've embedded all call to the + function in the call to the * function. The value from + is fed as data straight into the * function. *Expect to use this feature a lot.*

With entry 4, see how every argument after the first is subtracted from the first. We could have just as easily written **(- 10 (+ 5 5 5))** to get the same effect.

Finally, pay particular attention to entries 5 and 6, since they can be tricky. The type of result returned by the / function depends on its arguments. If integers are passed, and they don't divide cleanly, a fraction is returned. If they do divide cleanly, a regular integer is returned. However, if either argument is a floating point value such as 2.0 or 3.14, the result will always be a floating point value.

2) Basic List Handling

One of LISP's greatest strengths is its ability to operate on lists of information. The best way to get a handle on its many functions for doing this is to dive right in. Try running the following commands to see what they return:

1.) **(first '(1 2 3 4))**

2.) **(rest ‘(1 2 3 4))**
3.) **(first ())**
4.) **(rest ())**
5.) **(rest (first ‘(1 2 3))**

As you can see, first and rest are basically complementary to each other. Given a list of elements, first returns the first element. Last returns a list of all elements except for the first.

In cases where there is nothing to return, such as when used on an empty list, an empty list, or NIL, is returned.

One thing you might ask is why that single quote (‘) precedes some of the lists. To answer this, try entering the first command without it:

(first ‘(1 2 3 4))

You should get back something like this:

```
[2] CL-USER(17): (first (1 2 3 4))
Error: Funcall of 1 which is a non-function.
  [condition type: TYPE-ERROR]

Restart actions (select using :continue):
 0: Return to Debug Level 2 (an "abort" restart).
 1: Return to Debug Level 1 (an "abort" restart).
 2: Return to Top Level (an "abort" restart).
 3: Abort entirely from this (lisp) process.
[3] CL-USER(18):
```

The first line shows the offending command. The second line is the actual error. It says that it tried to call a function 1 but failed because 1 is not a function.

LISP always wants to evaluate the first element of a list as a function. We use the quotation mark to suppress this behavior. When we use the quotation mark here, we make it clear that the 1 is data, and not to be interpreted as a function.

To escape the above error, type :pop or :res, which respectively pop you out of the break stack or bring you out of it, returning you to the regular LISP listener.

3) Variables

Of course, like any full featured language, LISP allows us to bind values to names, creating variables. We do this with the setf special form. Type the following series of statements:

(setf brightred ‘(255 0 0))
brightred
(setf redvalue (first brightred))
redvalue
(setf greenvalue (first (rest brightred)))

greenvalue

Here we start by creating a list of numbers, which we assign to the variable brightred. We then start extracting values from this list and assigning them to other variables.

Note that typing the variable names by themselves wasn't necessary, but I want you to see that if you do so, it shows you the value of that variable.