CIS681 – Artificial Intelligence – Some Lisp Style Remarks

Just a word on what the functions you hand in should look like.

- Indent code to reflect level of nesting of parenthesis. Otherwise it is impossible to read – if you put that setenv thing in your .login file vi will do a pretty good job of getting the indentation right. Conventions:
- Make code reflect the way you think about the problem – recursively! Each function usually consists of a conditional checking base conditions and then a recursive call – usually cons the car onto result of doing function to rest of list.
- Variable and function names should be well-chosen (descriptive)
- Use special names for global variables (e.g., *data*).
- Use global variables only when they are the clearest way to do things (e.g., as pointers to a global data base).
- Stress functional embedding. Avoid temp or local variables. "The judicious use of LET and functional embedding can remove the need for most instances of SETQ. Doing this is considered the mark of an expert lisp programmer."
- Keep functions short. Break up large functions into several logically self-contained programs. This will make testing and debugging much easier!
- COMMENT CODE! Each function should have a block comment in the beginning explaining what kind of input arguments are expected, and what is returned. In the code, anything tricky should have a comment. Another mark of a good lisp programmer is to be able to look at a system written years before and figure out how it works in 5 minutes – find functions to borrow etc...

Testing Functions

When you hand in your code, you will want to show that the code actually works. The following function will help you and is included in the lisp-init.lisp file on the composers.

```
(defun print-eval (l)
  (mapcar (function (lambda (ele)
                      (print ele)
                      (terpri)
                      (print (eval ele))
                      (terpri)
                      (terpri)
                      ))
          l))

(print-eval '(
; put whatever expressions you want to test here
))
```