LISP Debugging Techniques

1. Comments are delineated by a semi-colon on a given line. This makes the LISP reader skip to the next line. Larger comments can be delineated by "hash-bar":

```
#| put comments here |#
```

In general the hash-mark, #, is a special token to the LISP reader, telling it that the token or object which follows is a special type.

2. Select the important functions in each program, and use TRACE, PRINT, and BREAK to follow the calls to that function. If recursive programs do not trace in your system, either rename the recursive function with an indirect function name, as in

```
(defun fac (n) (if (= n 1) 1 (* n (fac1 (- n 1)))))
(defun fac1 (n) (fac n))
```

or insert a not-in-line declaration immediately after the argument list in the function definition:

```
(defun fac (n)
      (declare (notinline fac))
      (if (= n 1) 1 (* n (fac (- n 1)))))
```

3. TRACE is a system function which traces and prints out each call to the specific traced function(s). Invoke it by evaluating

```
(trace <function-name>)
```

at the prompt or in the code. Disable it with

```
(untrace <function-name>)
```

or

```
(untrace)
```

for all traced functions. This invaluable tool lets you watch the dynamic calling patterns of functions.

4. Inserting a PRINT statement within the code will print a specific message when the code execution passes the point of the print function.

```
(print (list <item1> .. <itemn>)
```

5. Just like inserting a PRINT statement into a function to see the current bindings of local variables, you can insert

```
(break "message")
```

which temporarily halts execution at the point of insertion. The "message" portion is actually a FORMAT form. FORMAT is an elaborate print statement used for fine grain control of output. All you need to know, though, is that special FORMAT directives within the quoted message automatically bind to values of provided local variables. The directive is written "~A". Experiment with these functions:

```
(break "hello")
(break "the value of foo is ~A" <variable-named-foo>)
(break "values: ~A ~A ~A" <var1> <var2> <var3>)
```

Your system will let you continue from a BREAK with a special keystroke (my MCL-LISP uses *Apple-/*). So BREAK is a way of building custom stepping. I've put some examples of this tool in the Filters code package. It is customary to turn debug tools on and off by putting a comment character ";" in front of them textually and re-evaluating the function. Of course you could use a global debug flag if you like, like this

```
(defvar *debug-flag* nil)   ;put this at the top of the code package
      (when *debug-flag*
      (print 'foo) (break "foo-break"))  ;put this inline
```

A caution: Some of the more sophisticated code in the distribution may contain some built-in functions that may not work on your system. (A previous example of this was the LOOP construct in the ITERATIVE-UNIFY algorithm.) I've attempted to make sure that all code runs in vanilla environments, so bring any obstacles to class.