SCHEME SOLUTIONS TO ASSIGNMENTS

To access the Scheme solutions to Lisp Assignments 3, 4, and 5, you will need to use the kawa Scheme interpreter, which is available both on euclid and on venus. To start the kawa Scheme interpreter on euclid or venus, enter

scm

at the shell prompt. [On venus, this will only work if you followed the instructions regarding 316setup on the handout for Lisp Assignment 1.] You can exit scheme by entering (exit) at Scheme's prompt.

NOTE: Enter "scm" and NOT "scheme" to start the kawa Scheme interpreter. (If you enter "scheme", then you will start a different Scheme interpreter that is NOT able to read my solutions to the Assignments!)

The following table can be used to "translate" simple Scheme code (e.g., the Scheme code in my solutions to assignments or in Ch. 10 of Sethi) into Common Lisp:

Scheme	Common Lisp
(define x 3)	(setf x 3)
(define (f x y z) ...)	(defun f (x y z) ...)
#f [means "false"]	nil or ()
#t	t
() [the empty list]	nil or ()
else (as final guard of a COND)	t
null?	null
equal?	equal
even?	evenp
odd?	oddp
symbol?	symbolp
number?	numberp
integer?	integerp
pair?	consp
zero?	zerop

Note also that:

1. Whereas (car nil) and (cdr nil) return NIL in Common Lisp--which is illogical but occasionally convenient--(cdr ()) and (car ()) are undefined in standard Scheme.
2. () is NOT equivalent to #f in standard Scheme; () is a TRUE value! The symbol NIL is NOT equivalent to (), nor to #f; in fact NIL has no special meaning in Scheme.

Please also read the Technical Notes on the next page.

TECHNICAL NOTES

Lowercase letters in kawa Scheme symbol names are NOT automatically converted to upper case when they are read in. Most predefined functions and the functions defined in my solutions have lowercase names, and these names must be entered in lowercase when you call the functions.

Some of my solutions use a test (real? x). The Scheme predicate real? tests whether or not its argument is a real number, and is analogous to the predicate REALP in Common Lisp. As we are not using complex numbers in this course, I will not deduct points if you called NUMBERP instead of REALP.

My Scheme solution to the multiple-member problem uses a helping function safe-cdr that is like cdr but satisfies

```
(safe-cdr #f) => ()
```

The reason I used safe-cdr is that the Common Lisp solution I have in mind uses the fact that

```
(cdr nil) => nil
```

but Scheme is more logical than Common Lisp and produces an error when it evaluates (cdr #f). So I used safe-cdr as a Scheme analog of Common Lisp's cdr in this problem--in Common Lisp, you can just use cdr (or rest).