# Problem Set 1 Solutions

1. `(define x #t)`. Then `(* 1 x)` causes an error.

2. Let $M$ be

   ```
   (define x 0)
   (define zero-thunk (lambda() x))
   ((lambda(x) (zero-thunk)) 1)
   ```

   which evaluates to `0`. Instantiating `one-thunk` without renaming yields $N$:

   ```
   (define x 0)
   (define zero-thunk (lambda() x))
   ((lambda(x) ((lambda() x))) 1)
   ```

   which evaluates to `1`.

   Nitpick: the problem asked for Scheme expressions, but $M$ and $N$ above are bodies. To produce expressions with the requisite behavior, use `((lambda () `$M$`))` and `((lambda () `$N$`))`.

3. Cycle of rules is "ILIPCIP" for reflecting the repeated recursion until steps 33–41: (C,G) (I,G) P (I,G) P (I,G) P (I,G) P.

4. Operands in a combination are evaulated right-to-left in this implementation.

5. Actually the simplest example is a *self-evaluating* expression such as `#t` or `1`.

   Following the logic of the hint, we consider the "self-reproducing" expression

   ```
   (make-quoted-self-application 'make-quoted-self-application)
   ```

   developed in class. Now just use the body of the procedure instead of its name:

   ```
   ((lambda (s) (list s (list 'quote s))) '(lambda (s) (list s (list 'quote s))))
   ;Value 3:
   ((lambda (s) (list s (list 'quote s))) '(lambda (s) (list s (list 'quote s))))
   ```