Last Name: _____ *(as on your ID)*

First Name: _____

ID: _____ Score: / 25

Instructions: Write your solutions to the following problems in the space provided. Read each problem carefully and recheck your work. All work must be your own and you must work independently.

Q1 — REWRITING [5 POINTS]

The following procedure, `power-close-to`, finds the smallest power of its first argument that is greater than its second argument.

```
(define (power-close-to b n)
  (power-iter b n 1))

(define (power-iter b n e)
  (if (> (expt b e) n)
      e
      (power-iter b n (+ e 1))))
```

Embed the definition of `power-iter` inside `power-close-to` using the "named let" form. Take advantage of lexical scoping to remove unnecessary parameters from the embedded `power-iter`, and explain why you could remove those parameters.

Q2 — BOX-AND-POINTERS [5 POINTS]
Suppose we evaluate the expression

```
(list (cons 1 2) (list 3 4 '()))
```

Draw the corresponding box-and-pointer structure for the cons-cells generated.

Q3 — BOX-AND-POINTERS [5 POINTS]
Now suppose we evaluate the expressions, using the mutable variant of `cons` (`mcons`):

```
(define cell (mcons 1 3))
(set-mcar! cell (mcdr cell))
(set-mcdr! cell cell)
```

Draw the corresponding box-and-pointer structure for the (mutable) cons-cells generated.

Q4 — TAIL CALLS [5 POINTS]

Examine the five procedures below and mark if the algorithm is: **Recursive**, **Iterative** (i.e., all recursive calls are tail calls), or **Partially Iterative** (i.e., some recursive calls are tail calls, but not all).

A. `factorial`: **Recursive** **Iterative** **Partially Iterative**

```
(define (factorial n)
  (let fact ((i n) (a 1))
    (if (= i 0)
        a
        (fact (- i 1) (* a i)))))
```

B. `factor`: **Recursive** **Iterative** **Partially Iterative**

```
(define (factor n)
  (let f ((n n) (i 2))
    (cond ((>= i n) (list n))
          ((integer? (/ n i)) (cons i (f (/ n i) i)))
          (else (f n (+ i 1))))))
```

C. `list?`: **Recursive** **Iterative** **Partially Iterative**

```
(define (list? x)
  (let race ((h x) (t x))
    (if (pair? h)
        (let ((h (cdr h)))
          (if (pair? h)
              (and (not (eq? h t))
                   (race (cdr h) (cdr t)))
              (null? h)))
        (null? h))))
```

D. `even?`: **Recursive** **Iterative** **Partially Iterative**

E. `odd?`: **Recursive** **Iterative** **Partially Iterative**

```
(define (even? x)
  (or (= x 0)
      (odd? (- x 1))))

(define (odd? x)
  (and (not (= x 0))
       (even? (- x 1))))
```

Q5 — TYPING [5 POINTS]

For each of the following expressions, what type must `f` have in order for the evaluation of the expression to not cause an error? For each expression, give a definition of `f` such that evaluating the expression will not cause an error, and say what the expression's value will be, given your definition.

A. `f`

B. `(f)`

C. `(f 3)`

D. `((f))`

E. `(((f)) 3)`