Quiz 8a Rubric

1. (3 points) Fill in the blanks with the values of the expressions shown:

(define x 0)

(define mystery

(let ((x x)

(y 0))

(lambda ()

(set! y (+ y 1))

(set! x (+ x y))

x)))

(mystery) \_\_\_\_1\_\_\_

(mystery) \_\_\_\_3\_\_\_

(mystery) \_\_\_\_6\_\_\_

x \_\_\_\_0\_\_\_

-1 for every incorrect response.

2.(4 points) Write procedure msc (short for “make strange counter”) that returns a counter procedure. The first counter msc returns is a counter that starts from 1 and counts up by 1 at a time. Later on, when msc is called, it returns a counter that starts from k and counts up by k, where k is the largest number ever returned by *any* strange counters so far.

Examples:

STk> (define c1 (msc)) ; msc is called for the first time

c1 ; c1 is a counter starting from 1, counting by 1

STk> (c1)

1

STk> (c1)

2

STk> (c1)

3

STk> (define c2 (msc))

c2 ; c2 is a counter starting from 3, counting by 3

STk> (c2)

3

STk> (c2)

6

STk> (c1) ; note it's c1 here

4

STk> (define c3 (msc))

c3 ; c3 is a counter starting from 6, counting by 6

(define msc ; note no parentheses around msc!

(let ((big 1)) ; BIG is a class variable

(lambda () ; this is the procedure MSC itself

(let ((skip big)

(count 0)) ; these are instance variables

(lambda () ; this is a particular counter procedure

(set! count (+ count skip))

(if (> count big) (set! big count))

count)))))

-1 for not having “big” be a “class” variable.

-1 for not set!ing “big” if count becomes bigger than “big”

-1 for not having instance variables.

-1 for any other non-trivial syntax or logical error

3. (3 points)

Draw the environment diagram resulting from evaluating the following expressions, and show the result printed by the last expression where indicated.

> (define foo

(let ((var 5))

(define (max! var new)

(set! var (max var new))

var)

(lambda (x)

(define y (\* 2 x))

(max! y var)))))

> (foo 3)

Here are the steps in drawing the diagram.

1. The first thing evaluated is the implicit lambda from the LET.

This creates procedure P1, parameter VAR, body (DEFINE ...) (LAMBDA ...)

[two expressions], right bubble pointing to the global environment.

2. The next step is to evaluate the actual argument of the LET, namely

the self-evaluating number 5. This doesn't change the diagram.

3. Then we apply P1 to 5 by creating frame E1, with the binding VAR=5,

extending the global environment (because that's where P1's right bubble

points). In the new environment E1 we can now evaluate P1's body.

4. The first expression in the body is the (DEFINE (MAX! ...) ...).

We start with the implicit lambda in that definition, creating procedure

P2 with parameters VAR and NEW, body (SET! ...) VAR, and right bubble

pointing to the current environment, E1.

5. Now we finish the DEFINE, by making a binding from symbol MAX! to

procedure P2, adding that binding to the current frame E1.

6. Now we're ready for the second expression in the body of the LET,

namely (LAMBDA (X) ...). This creates procedure P3 with parameter X,

body (DEFINE Y ...) (MAX! ...), and right bubble pointing to E1.

7. Since that lambda expression was the last thing in the body of P1,

procedure P3 is the return value from P1. So we now finish evaluating

the top-level expression (DEFINE FOO ...) by adding a binding from FOO

to P3 in the global environment.

8. Now we evaluate (FOO 3). Evaluating the subexpressions FOO and 3

doesn't change the environment. Now we apply FOO, that is, procedure P3,

to the actual argument value 3. To do that, we make frame E2, with a

binding from X to 3, extending E1, because E1 is in P3's right bubble.

9. In environment E2, we evaluate the body of P3. The first expression

is a DEFINE; we evaluate (\* 2 X) -- which works because E2, the current

environment, has a binding for X -- and get 6, then we add a binding to

the current frame (E2) from Y to 6.

10. The remaining expression is (MAX! Y VAR). The current environment

has bindings for all these symbols, Y in frame E2 and the other two in frame

E1, which is part of environment E2. The value of MAX! is P2; the value of

Y is 6; the value of VAR is 5. So we can apply P2 to 6 and 5, by making

frame E3 with bindings VAR=6, NEW=5, extending E1 -- not E2, as it would be

in dynamic scope -- because P2's right bubble points to E1.

Note that E3's VAR is different from E1's VAR.

11. Now we evaluate the body of MAX! (that is, of P2) with E3 as current

environment. The SET! changes the binding of VAR \*in the current frame\*,

which is E3, so in fact there's no change, since that VAR is already 6 and

the new value is also 6. (Since MAX, as opposed to MAX!, is a primitive,

the evaluation of (MAX VAR NEW) doesn't change the diagram.)

12. Finally, we return VAR, which is 6.

Scoring:

3 Correct.

2 The SET! changes E1's VAR instead of E3's VAR.

2 The arguments to MAX! are swapped (VAR=5, NEW=6).

1 Dynamic scope (E3 extends E2).

1 The \*arguments\* to MAX! are evaluated in E1, so VAR=5 but Y is unbound.

(In this case, the final result should be ERROR instead of 6.)

1 The arguments to MAX! aren't evaluated at all, so in frame E3 we have

variable VAR bound to the symbol Y and/or NEW bound to the symbol VAR.

1 The (DEFINE Y ...) is treated as if it were (DEFINE (Y) ...), so Y is

bound to a procedure.

0 Even worse.