Quiz 2a Rubric

1. (2 points) The following two expressions are typed into Scheme in this order:

(define a 10)

(let ((a 5)(b a))

(< a b))

* 1. Expand the let expression into the lambda-based expression

that it abbreviates.

((lambda (a b) (< a b)) 5 a)

* 1. What value does the let expression return?

#t

One point for each all or nothing

1. (4 points) Write a procedure alternator that takes two one-argument procedures

(we'll call them F and G) as its argument. It

returns a procedure that takes a sentence as its argument, and applies

F to the odd-numbered words of the sentence, and G to the even-numbered

words, returning a sentence of the results:

> ((alternator square (lambda (x) (+ x 1))) '(3 4 5 6 7))

(9 5 25 7 49)

> (define filast (alternator first last))

> (filast '(do you want to know a secret))

(d u w o k a s)

(define (alternator F G)

(lambda (sent)

(if (empty? sent)

‘()

(se (F (first sent)) ((alternator G F) (bf sent))))))

1 point – evaluates to a lambda

1 point – arguments are tow functions

2 points – recursion is correct

1. (4 points) Write a procedure do-n that takes three arguments: a function func

of one argument, a sentence args of data, and a sentence times of numbers.

Your procedure will take each element of args, and apply func to

that element, then again to the result, then to *that* result, etc.,

repeating the use of func the number of times given in the corresponding

element of the sentence times. You may assume the two sentences are the same

length. You may further assume the value returned by func will be a

valid argument to that function.

You may write and use helper procedures.

For example:

> (do-n square '(1 2 3) '(9 3 2)) ;2nd number in result is 256 because

(1 256 81) ; 22=4; 42=16; 162=256

> (do-n butlast '(twinkle twinkle little star) '(2 5 4 1))

(twink tw li sta)

> (do-n cdr '((mary had a little lamb) (with fleece as white as snow))

'(2 3))

((a little lamb) (white as snow))

}

Start with this:

(define (do-n func args times)

(define (n-times func arg times)

(if (= times 0)

arg

(func (n-times func arg (- times 1)))))

(if (empty? args)

‘()

(se (n-times func (first args) (first times)) (do-n func (bf args) (bf times)))))

4 points – correct

3 points – mostly correct one or two minor errors

2 points – Program structure is correct but not functional

1 point – Part of student’s answer matches solution