CSSE304 Exam 1 Paper Review

Instructions

Please complete the following problems. If you have questions come up and ask. I will send out an answer key after the review session is over.

1. **Problem 1:** Write a function modify-a-list that takes single parameter scheme procedure and returns a procedure that takes a list as an argument. Hint: think about map and apply.

```
(define inc-by-1 (modify-a-list add1))

(inc-by-1 '(1 2 3)) -> '(2 3 4)
(inc-by-1 1 2) -> Error
((modify-a-list list)'(a b)) -> '((a) (b))
```

```
(define modify-a-list
  (lambda (proc)
    (lambda (x)
      (map proc x))))
```

2. **Problem 2:** Write a function product-of-evens that takes in a list of integers and returns the product of only the even values. Use some combination of map, apply, and filter to complete the problem (no explicit recursion).

```
(product-of-evens '(1 2 3 4 5)) -> 8
```

```
(define product-of-evens
  (lambda (x) (apply * (filter even? x))))
```

3. **Problem 3:** Write a function product-incremented-numbers that takes in a list of values (numbers and symbols) and returns the product of only the numbers incremented by 1. Use some combination of map, apply, and filter to complete the problem (no explicit recursion).

```
(product-incremented-numbers '(1 a 3 4 f s)) -> 40
```

```
(define product-incremented-numbers
  (lambda (x)
    (apply * (map add1 (filter number? x)))))
```

4. **Problem 4:** Here is some code that uses let and lambda to show closures. Write what will be displayed by the code when it is run.

```
(define factory
  (let ((global-val 100))
    (lambda ()
      (let ((local-val 0))
        (lambda ()
          (let ((temp-val local-val))
            (set! global-val (- global-val 5))
            (set! local-val (+ local-val 10))
            (set! temp-val (+ temp-val 1))
            (display (list global-val local-val temp-val))
            (newline)))))))

(define f1 (factory))
(define f2 (factory))
(f1)
(f1)
(f2)
(f1)
```

(95 10 1)
(90 20 11)
(85 10 1)
(80 30 21)

5. **Problem 5:** Here is a lambda calculus expression. list which values are bound and which are free.

```
(lambda (x) (lambda (y) (+ x y z)))
```

bound: x, y

free: z

6. **Problem 6:** Use the following grammar and expression to create a grammar tree. Use E for expr, B for base_val, S for symbol, and I for integer.

```
<expr> ::= (<symbol> <expr> <expr>) | [<expr> <expr>]
         | <base_val>
<base_val> ::= <integer> | <symbol>
```

(+ a [3 (* 4 5)]))