# 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.

## Conceptual Questions

1. **Problem 1:** Describe the Racket programming language and explain some of its use cases.

2. **Problem 2:** Describe a meta-function and give an example of a use case of one.

3. **Problem 3:** What is a first class procedure?

4. **Problem 4:** What does it mean if a procedure is "Short Circuiting"?

5. **Problem 5:** How is Racket scoped? Explain what that means in terms of running code.

# Code Based Questions

1. **Problem 1:** Create the contour diagram for the following code snippet:

   ```
   (define silly-proc
     (lambda (a b c)
       (let ([d (* a c)])
         (lambda (e)
           (let ([a (let ([a (+ d a e)]
                          [f 11])
                      (+ a f c))])
             (+ e a c))))))
   ```

2. **Problem 2:** What does running (silly-proc 1 2 3) return?

3. **Problem 3:** Evaluate the following code. Show the values stored in each let and lambda expression.

   ```
   (define silly-proc
     (lambda (a b c)
       (let ([d (* a c)])
         (lambda (e)
           (let ([a (let ([a (+ d a e)]
                          [f 11])
                      (+ a f c))])
             (+ e a c))))))

   ((silly-proc 3 4 5) 2)
   ```

4. **Problem 4:** Convert the following let-expression to a lambda-application expression and evaluate its output.

   ```
   ((lambda (x)
     (let ([a (+ x 1)])
       (let ([b (+ a 2)])
         ((lambda (y)
           (+ a b y)) b)))) 5)
   ```

5. **Problem 5:** Write a function find-valid-data that takes in a list of lists of numbers and returns a list of only the positive averages. Use some combination of map, apply, and filter to complete the problem (no explicit recursion).

```
(find-valid-data '((1 2 3) (-2 -4 -2) (0 3 5 4))) -> '(2 3)
```

6. **Problem 6:** 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 5))
            (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)
```

7. **Problem 7:** 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)]))
```