**Structures and Interpretation of Computer Program**

**Exercise Chapter 1.3 Name:** Wan Huzaifah bin Wan Azhar

**Exercise 1.3.4 Procedures as Returned Value**



(define (cube x) (\* x x x))

(define (square x) (\* x x))

(define (cubic a b c)

(lambda (x)

(+ (cube x)

(\* a (square x))

(\* b x) c)))



(define (inc x) (+ x 1))

(define (double f)

(lambda (x)

(f (f x))))

(display ((double inc) 5))

The procedure (((double (double double)) inc) 5) is equal to (+ x 16). By evaluating (double double) with inc argument, it returns an equivalent of (+ x 8). Double of (+ x 8) is (+ (+ x 8) 8) which is equal to ((+ x 16). As x = 5, the final value is 21



(define (compose f g)

(lambda (x)

(f (g x))))



(define (repeated f n)

(lambda (x)

(if (= 1 n)

(f x)

((compose (repeated f (- n 1)) f) x))))

(define (square x) (\* x x))

(display ((repeated square 3) 5))



(define (square x) (\* x x))

(define (smooth f)

(lambda (x)

(/ (+ (f (- x dx))

(f x)

(f (+ x dx)))

3)))

(define (smoothing f n)

((repeated smooth n) f))

(display ((smooth square) 2))

(newline)

(display ((smoothing square 10) 2))



(define (log2 x) (/ (log x) (log 2)))

(define (nth-root n x)

(let ((k (- n 1)))

(fixed-point

((repeated average-damp (floor (log2 n)))

(lambda (y) (/ x (expt y k))))

1.0)))

(display (nth-root 6 729))

Calculation for how many average damp is needed is the floor of n. For example, floor of log of 9 is 3, 7 is 2. Here I use log2 instead.

To be honest, I cheat a bit on this exercise as my code cannot converge when rooted by 6. Turns out it was because my code calculates the repeat of applied x on average damp first instead of calculating the average damp formula first before applying x. Which in turns, takes a long time to calculate.



(define (cube-root x)

(fixed-points

(repeated (average-damp (lambda (y) (/ x (expt y 2)))) 2)

1.0 ))

(define (iterative-improve good-enough? improve-guess)

(define (try guess)

(let ((next (improve-guess guess)))

(if (good-enough? guess next)

next

(try next))))

(try 1.0))

(define (fixed-points f first-guess)

(iterative-improve

(lambda (v1 v2) (< (abs (- v1 v2)) 0.0001))

(lambda (x) (f (f x)))

))

(define (sqrt-iter x)

(iterative-improve

(lambda (guess y) (< (abs (- (square guess) x)) 0.001))

(lambda (guess) (average guess (/ x guess)))

))

(define (sqrt x)

(sqrt-iter x))

(display (cube-root 27))

(newline)

(display (sqrt-iter 4))

Output:

2.9999972321057697

2.000000000000002