**Structures and Interpretation of Computer Program**

**Exercise Chapter 2.2 Name:** Wan Huzaifah bin Wan Azhar

**Exercise 2.2.2 Sequences as Conventional Interfaces**



(define (map p sequence)

(accumulate (lambda (x y) (cons (p x) y)) nil sequence))

(define (append seq1 seq2)

(accumulate cons seq2 seq1))

(define (length sequence)

(accumulate (lambda (x y) (+ 1 y)) 0 sequence))



(define (horner-eval x coefficient-sequence)

(accumulate (lambda (this-coeff higher-terms) (+ this-coeff (\* x higher-terms)))

0

coefficient-sequence))



(define (count-leaves t)

(accumulate (lambda (x y) (+ 1 y)) 0 (enumerate-tree t)))



(define (accumulate-n op init seqs)

(if (null? (car seqs))

nil

(cons (accumulate op init

(accumulate (lambda (x y) (cons (car x) y)) nil seqs))

(accumulate-n op init

(accumulate (lambda (x y) (cons (cdr x) y)) nil seqs)))))

2. Matrix-Vector Multiplication

(define (dot-product v w)

(accumulate + 0 (map \* v w)))

(define (matrix-\*-vector m v)

(map (lambda (row) (dot-product row v)) m))

(define matrix-a (list (list 1 2 3) (list 3 4 5) (list 5 6 7)))

(define vector-a (list 1 2 3))

(display (matrix-\*-vector matrix-a vector-a))

Output:

(14 26 38)

1. Transposing Matrix

(define (transpose mat)

(accumulate-n cons nil mat))

1. Matrix-Matrix Multiplication

(define (matrix-\*-matrix m n)

(let ((cols (transpose n)))

(map (lambda (row) (matrix-\*-vector cols row)) m)))

(define matrix-b (list (list 1 2) (list 3 4)))

(display (matrix-\*-matrix matrix-b matrix-b))

Output:

((7 10) (15 22))



* (fold-left / 1 (list 1 2 3)) = (/ 1 2 3) = 1.6667
* (fold-right / 1 (list 1 2 3)) = (/ 1 (/ 2 (/ 3 1))) = (/ 1 (/ 2 3) = (/ 3 2) = 1.5
* (fold-right list nil (list 1 2 3)) = (1 (2 (3)))
* (fold-left list nil (list 1 2 3)) = (((nil 1) 2) 3)
* The op should be commutative so that operation from the left or right will produce the same result.
  + An example is + or \*



**Nested Mappings**



(define (prime-sum-pairs n)

(map make-pair-sum

(filter prime-sum?

(unique-pairs n))))

(define (unique-pairs n)

(flatmap

(lambda (i)

(map (lambda (j) (list i j))

(enumerate-interval 1 (- i 1))))

(enumerate-interval 1 n)))



(define (unique-triples n)

(flatmap

(lambda (k)

(map (lambda (y) (cons k y))

(unique-pairs n)))

(enumerate-interval 1 n)))

(display (unique-triples 3))

Output:

((1 2 1) (1 3 1) (1 3 2) (2 2 1) (2 3 1) (2 3 2) (3 2 1) (3 3 1) (3 3 2))