**Structures and Interpretation of Computer Program**

**Exercise Chapter 2.1 Name:** Wan Huzaifah bin Wan Azhar

**Exercise 2.1.3 What is meant by data**



(define one-four (list 1 2 3 4))

(define (last-pair list)

(if (null? (cdr list))

(car list)

(last-pair (cdr list))))

(display (last-pair one-four))

Output:

4



(define two-five (list 1 2 3 4 5 6 7))

(define (reversal list1)

(define list2 '())

(define (reversal-iter list1 list2)

(if (null? (cdr list1))

(cons (car list1) list2)

(reversal-iter (cdr list1) (cons (car list1) list2))))

(reversal-iter list1 list2))

(display (reversal two-five))

Output:

(7 6 5 4 3 2 1)



(define us-coins (list 50 25 10 5 1))

(define uk-coins (list 100 50 20 10 5 2 1 0.5))

(define myr-coins (list 100 50 20 10 5))

(define (ccs amount coin-values)

(cond ((= amount 0) 1)

((or (< amount 0) (no-more? coin-values)) 0)

(else

(+ (ccs amount

(except-first-denomination coin-values))

(ccs (- amount

(first-denominations coin-values))

coin-values)))))

(define (first-denominations coin-values)

(car coin-values))

(define (except-first-denomination coin-values)

(cdr coin-values))

(define (no-more? coin-values)

(if (null? coin-values)

#t

#f

))

(display (ccs 100 us-coins))



(define (even? x) (= 0 (remainder x 2)))

(define (odd? x) (not (even? x)))

(define (generate-list list1 op)

(cond ((null? list1) '() )

((op (car list1)) (cons (car list1) (generate-list (cdr list1) op)))

(else (generate-list (cdr list1) op))))

(define (same-parity . list)

(if (even? (car list))

(generate-list list even?)

(generate-list list odd?)))

(display (same-parity 1 2 3 4 5 7 8 9))

(newline)

(display (same-parity 2 3 4 5 7 8 9))

Output:

(1 3 5 7 9)

(2 4 8)

Mapping over list



(define (map proc items)

(if (null? items)

'()

(cons (proc (car items))

(map proc (cdr items)))))

(define (square-list items)

(define (sqr x) (\* x x))

(if (null? items)

'()

(cons (sqr (car items)) (square-list (cdr items)))))

(define (square-list2 items)

(map (lambda (x) (\* x x)) items))

(display (square-list2 (list 1 2 3 4)))

2. This is because Louis Reasoner append things in reverse.
   1. As it can be seen it (cons (square car things) answer)
   2. If the list given is 1 2 3 4
   3. It first appends 1^2 nil
   4. Then 2^2 1^2 nil
   5. And so on..
3. This doesn’t work because cons are now defined inside itself
   1. For example, given list 1 2 3 4
   2. (cons nil 1^2)
   3. (cons (cons nil 1^2) 2^2)
   4. (cons (cons (cons 1^2) 2^2) 3^2) and so on..
   5. It now become inner chains of cons.

(define (even? x) (= 0 (remainder x 2)))

(define (odd? x) (not (even? x)))

;Uses its argument to apply procedure

(define (for-each proc items)

(define (for-each-iter proc apply-proc items)

(if (null? (cdr items))

(proc (car items))

(for-each-iter proc (proc (car items)) (cdr items))))

(for-each-iter proc 0 items))

(for-each (lambda (x) (newline) (display x))

(list (odd? 57) 321 88))

Output:

#t

321

88