**Structures and Interpretation of Computer Program**

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

**Exercise 1.3.1 Procedures as Arguments**

1. Create procedure integral using Simpson rule and compare with textbook’s integral

(define (sum term a next b)

(if (> a b)

0

(+ (term a)

(sum term (next a) next b))))

(define (integral f a b dx)

(define (add-dx x) (+ x dx))

(\* (sum f (+ a (/ dx 2.0)) add-dx b)

dx ))

(define (even? x)

(= 0 (remainder x 2)))

(define (h a b n)

( / (- b a) n))

(define (y k f h a)

(f (+ a (\* k h))))

(define (integrate a b f n)

(\* (/ (h a b n) 3) (integrate-rec a b f n 0))

)

(define (integrate-rec a b f n k)

(cond ((> k n) 0)

((or (= k n) (= k 0))

( + (y k f (h a b n) a) (integrate-rec a b f n (+ k 1))))

((even? k)

( + (\* 2 (y k f (h a b n) a)) (integrate-rec a b f n (+ k 1))))

(else

( + (\* 4 (y k f (h a b n) a)) (integrate-rec a b f n (+ k 1))))

))

(define (cube x )

(expt x 3)

)

(display (integrate 0 1.0 cube 100))

(newline)

(display (integral cube 0 1 0.001))

(newline)

(display (integrate 0 1.0 cube 1000))

(newline)

(display (integral cube 0 1 0.0001))

Output:

0.24999999999999992

0.249999875000001

0.2500000000000002

0.24999999874993412



(define (sum term a next b)

(define (iter a result)

(if (> a b)

result

(iter (inc a) (+ (term a) result))))

(iter a 0))

1. Recursive product

(define (product term a next b)

(if (> a b)

1

(\* (term a)

(product term (next a) next b))))

(define (factorial x)

(product identity 1 inc x ))

(define (even? x)

(= 0 (remainder x 2)))

(define (product term a next b)

(if (> a b)

1

(\* (term a)

(product term (next a) next b))))

(define (inc x)

(+ x 1))

(define (pi-over-four n)

(product pi-term 1 inc n))

(define (pi-term n)

(if (even? n)

(/ (+ n 2) (+ n 1))

(/ (+ n 1) (+ n 2))))

(display (pi-over-four 3))

1. Iterative product

(define (product-iter term a next b)

(define (iter a result)

(if (> a b)

result

(iter (next a) (\* (term a) result))))

(iter a 1))

(define (pi-over-four n)

(accumulate \* 1 pi-term 1 inc n))

(define (pi-term n)

(if (even? n)

(/ (+ n 2) (+ n 1))

(/ (+ n 1) (+ n 2))))

(define (accumulate combiner null-value term a next b)

(if (> a b)

null-value

(combiner (term a)

(accumulate combiner null-value term (next a) next b))))

(display (pi-over-four 3))



(define (accumulate-iter combiner null-value term a next b)

(define (iter a result)

(if (> a b)

result

(iter (next a) (combiner (term a) result))))

(iter a null-value))

(display (pi-over-four 3))

2. Accumulator filter prime

(define (filtered-accumulate combiner filter null-value term a next b)

(cond ((> a b)

null-value)

((filter a)

(combiner (term a)

(filtered-accumulate combiner filter null-value term (next a) next b)))

(else

(combiner null-value

(filtered-accumulate combiner filter null-value term (next a) next b)))))

(define (sum-of-prime a b)

(filtered-accumulate + prime? 0 identity a inc b))

1. Accumulator filter GCD(I, n) = 1

(define (gcd? a b)

(if (= b 0)

a

(gcd b (remainder a b))))

(define (product-less-than-n-prime n)

(define (less-than-and-relprime x)

(and (< x n) (= 1 (gcd? x n))))

(filtered-accumulate \* less-than-and-relprime 1 identity 1 inc n))

(display (product-less-than-n-prime 10))