**Structures and Interpretation of Computer Program**

**Exercise Chapter 2.3 Name:** Wan Huzaifah bin Wan Azhar

**Exercise 2.3.2 Representing Algebraic Expression**

(define (exponentiation? x)

(and (pair? x) (eq? (car x) '\*\*)))

(define (base s) (cadr s))

(define (exponent s) (caddr s))

(define (make-exponentiation base expnt)

(cond ((=number? expnt 0) 1)

((=number? expnt 1) base)

(else (list '\*\* base expnt))))

(define (deriv exp var)

(cond ((number? exp) 0)

((variable? exp)

(if (same-variable? exp var) 1 0))

((sum? exp)

(make-sum (deriv (addend exp) var)

(deriv (augend exp) var)))

((product? exp)

(make-sum

(make-product (multiplier exp)

(deriv (multiplicand exp) var))

(make-product (deriv (multiplier exp) var)

(multiplicand exp))))

((exponentiation? exp)

(make-product ;chain make-product as it can only take 2 argument

(make-product (exponent exp)

(make-exponentiation (base exp)

(make-sum (exponent exp) -1)))

(deriv (base exp) var)))

(else

(error "unknown expression type -- DERIV" exp))))



(define (multiplicand p)

(let ((first (caddr p))

(rest (cdddr p)))

(if (null? rest)

first

(make-product first (cadddr p)))))

(define (augend s)

(if (null? (cdddr s))

(caddr s)

(make-sum (caddr s) (cadddr s))))



(define (make-sum a1 a2)

(cond ((=number? a1 0) a2)

((=number? a2 0) a1)

((and (number? a1) (number? a2)) (+ a1 a2))

(else (list a1 '+ a2))))

(define (sum? x)

(and (pair? x) (eq? (cadr x) '+)))

(define (addend s) (car s))

(define (augend s) (caddr s))

(define (make-product m1 m2)

(cond ((or (=number? m1 0) (=number? m2 0)) 0)

((=number? m1 1) m2)

((=number? m2 1) m1)

((and (number? m1) (number? m2)) (\* m1 m2))

(else (list m1 '\* m2))))

(define (product? x)

(and (pair? x) (eq? (cadr x) '\*)))

(define (multiplier p) (car p))

(define (multiplicand p) (caddr p))

(display (deriv '(x + (3 \* (x + (y + 2)))) 'x))