**Jonathan Julian**

**CSc600**

**12/10/19**

**Scheme HW**

**1A.** > ((lambda (x) (+ x x)) 3)

6

**B.** > (define add1 (lambda (x) (+ x 1)))

> (add1 10)

11

**C.** > (define funlist '(add1 sub1))

> funlist

'(add1 sub1)

**D.** > (equal? add1 add1)

#t

> (eq? add1 add1)

#t

> (eqv? add1 add1)

#t

> (eq? list1 list1)

#t

> (eqv? list1 list1)

#t

> (equal? list1 list1)

#t

> (define list2 '(1 2 3))

> (equal? list1 list2)

#t

> (eqv? list1 list2)

#f

> (eq? list1 list2)

#f

**E.** > (abs (add1 -9))

8

**F.** > (define (adding n) (lambda (x) (+ n x)))

> ((adding 10) 2)

12

**G.** > (define inport (open-input-file "scheme1.dat"))

> (read inport)

'Scheme

> (read inport)

'input

> (read inport)

#<eof>

**2.**

(define sigma (lambda args

(let\* ([n (length args)]

[xbar (/ (sumList args) n)]

[x2bar (/ (squareList args) n)])

(sqrt (- x2bar (\* xbar xbar))))))

(define sumList (lambda (list)

(if (null? list)

0

(+ (car list) (sumList (cdr list))))))

(define squareList (lambda (list)

(if (null? list)

0

(+ (\* (car list) (car list)) (squareList (cdr list))))))

> (sigma 1 3)

1

> (sigma 1 3 1 3 1 3)

1

> (sigma 1 2 3 2 1)

0.7483314773547883

**3 A.**

(define line (lambda (x)

(cond

((<= x 0) (newline))

(else

(display '\*)

(line (- x 1))

))))

> (line 5)

\*\*\*\*\*

> (line 10)

\*\*\*\*\*\*\*\*\*\*

**B.**

(define histogram (lambda args

(cond

((null? args) (values)) ;(values) is the NON printable object substitute. This is base case

(else

(line (car args)) ; call to line

(apply histogram (cdr args))) ;apply unwraps the nested list

)))

> (histogram 1 2 3 3 2 1)

\*

\*\*

\*\*\*

\*\*\*

\*\*

\*

**4.**

(define displayCoord (lambda (x n)

(display (/ (round (\* x (expt 10 n))) (expt 10 n)))))

(define fmax (lambda (f x1 x2)

(cond((< (- x2 x1) 1.e-6)

(display "maximum: f(")

(displayCoord (/ (+ x1 x2) 2) 6)

(display ") = ")

(displayCoord (f (/ (+ x1 x2) 2)) 6))

(else (let ((a1 (+ x1 (/ (- x2 x1) 3)))

(a2 (- x2 (/ (- x2 x1) 3))))

(if (< (f a1) (f a2))

(fmax f a1 x2)

(fmax f x1 a2)))))))

> (fmax cos -1 1)

maximum: f(0) = 1.0

> (fmax (lambda (x) (/ 1 (\* x x))) .1 100)

maximum: f(0.1) = 99.999207

**5 A.**

(define (scalar-product vector1 vector2)

[cond

[(eq? (vector-length vector1)(vector-length vector2))

( do (( index 0 (+ 1 index))(sum 0))

((>= index (vector-length vector1)) sum)

(set! sum (+ sum (\* (vector-ref vector1 index) (vector-ref vector2 index)))))]

[else (display "ERROR: Different sizes of vectors! ")]

])

> (scalar-product '#(1 2 3) '#(1 2 3 4 5))

ERROR: Different sizes of vectors!

> (scalar-product '#(1 2 3) '#(2 1 1))

7

**B.** (define (scalar-product vector1 vector2)

[cond

[(eq? (vector-length vector1)(vector-length vector2))

(display (scalar-help vector1 vector2 (- (vector-length vector1) 1)))]

[else (display "ERROR: Different sizes of vectors! ")]

])

(define (scalar-help vector1 vector2 index)

[cond

[(< index 0)

0]

[else (let

([sum (\* (vector-ref vector1 index) (vector-ref vector2 index))])

(+ sum (scalar-help vector1 vector2 (- index 1))))]])

> (scalar-product '#(1 2 3) '#(2 1 1))

7

> (scalar-product '#(1 2 3) '#(2 1 1 1))

ERROR: Different sizes of vectors!

**6A.**

(define (row fileName rowNumber)

(define in (open-input-file fileName))

(define row (read in))

(define column (read in))

(define totalElements (\* row column))

(define elementEnd (\* column rowNumber))

(define elementBegin (+ (\* column (- rowNumber 1)) 1))

(define vector (make-vector column))

(cond

[(> rowNumber row) (display "ERROR: row desired is higher than row count")]

[else

(do ((index 1 (+ index 1))(desiredIndex rowNumber)(vectorIndex 0))

((> index totalElements) (display vector))

(cond

[ (and (>= index elementBegin) (<= index elementEnd))(vector-set! vector vectorIndex (read in))

(set! vectorIndex (+ 1 vectorIndex))]

[else (read in)]

)

)]))

(define (col fileName colNumber)

(define in (open-input-file fileName))

(define row (read in))

(define column (read in))

(define totalElements (\* row column))

(define vector (make-vector row))

(cond

[(> colNumber column) (display "ERROR: column desired is higher than column count")]

[else

(do ((index 1 (+ index 1)) (desiredIndex colNumber)(vectorIndex 0))

((> index totalElements) (display vector))

(cond

[(= index desiredIndex) (set! desiredIndex (+ desiredIndex column)) (vector-set! vector vectorIndex (read in))

(set! vectorIndex (+ 1 vectorIndex))]

[else (read in)]

)

)]))

> (col "matrix1.dat" 1)

#(1 4)

> (col "matrix1.dat" 2)

#(2 5)

> (col "matrix1.dat" 3)

#(3 6)

> (col "matrix1.dat" 4)

ERROR: column desired is higher than column count

> (row "matrix1.dat" 1)

#(1 2 3)

> (row "matrix1.dat" 2)

#(4 5 6)

> (row "matrix1.dat" 3)

ERROR: row desired is higher than row count

**B.**

(define (matrix-row-number matrixfile) ;Helper function to get number of rows

(define port (open-input-file matrixfile))

(define row-number (read port))

(begin (close-input-port port) row-number))

(define (matrix-col-number matrixfile) ;Helper function to get number of cols

(define port (open-input-file matrixfile))

(define col-number '())

(begin (read port) (set! col-number (read port)) col-number))

(define (mmul inputfile1 inputfile2 outputfile)

(define port (open-output-file outputfile #:exists 'replace))

(begin

(display (vector-length (col inputfile1 1)) port) (display " " port)

(display (vector-length (row inputfile2 1)) port) (display "\r\n" port)

(do ((r 1 (+ 1 r)))

((> r (matrix-row-number inputfile1)))

(begin

(do ((c 1 (+ 1 c)))

((> c (matrix-col-number inputfile2)))

(begin (display (scalar-product (row inputfile1 r) (col inputfile2 c))) (display " ")

(display (scalar-product (row inputfile1 r) (col inputfile2 c)) port) (display " " port))

(display "")))

(newline) (display "\r\n" port))

(close-output-port port)))

(define (scalar-product vector1 vector2)

[cond

[(eq? (vector-length vector1)(vector-length vector2))

(scalar-help vector1 vector2 (- (vector-length vector1) 1))]

[else (display "ERROR: Different sizes of vectors! ")]

])

> (mmul "matrix1.dat" "matrix2.dat" "matrix3.dat")

6 12 18

15 30 45