**David Tran**

**CSC 600**

**Scheme**

**4/24/17**

**1. The concept of first class objects is fundamental for Scheme programming. In particular, in**

**Scheme functions are first class objects. The main properties of functions as first class objects**

**are exemplified by answering the following questions:**

#lang racket

#|

Author: David Tran

File: firstClass.rkt

Purpose: To practice and see examples of first class language

Source: Dr. Jozo Dujmovic Programming Languages reader (scheme section).

|#

**; a) Example of anonymous function**

(display "Anonymous Function\n")

((lambda (n) (\* n n)) 5)

**; b) Example of a named function**

(display "\nNamed Function\n")

(define square (lambda(n) (\* n n)))

(square 5)

**; c) Example of creating a list of functions and displaying the list**

(display "\nFunction List\n")

(define cube (lambda(n) (\* n n n)))

(define list '(square cube))

(display list)

**; d) Example of comparing a list of functions and comparing functions using equal? & eq?**

(display "\n\nCompare List of Functions\n")

(define list2 '(cube square))

(define list3 '(square cube))

(display list)

(display list2)

(display list3)

(display "\n")

(equal? list list2)

(equal? list list3)

(eq? square cube)

(eq? square square)

**; e) Example of passing functions to another**

(display "\nPassing Functions\n")

(cube(square(square 2)))

(cube(square(cube(square 2))))

**; f) Example of returning a function from another function using conditions**

(display "\nReturning a function as a result of another function\n")

(define mult2 (lambda(n) (\* n 2)))

(define mult3 (lambda(n) (\* n 3)))

(define multFunction (lambda(x y)

(cond

[(equal? x 1) (mult2 y)]

[(equal? x 2) (mult3 y)])))

(multFunction 1 2)

(multFunction 2 2)

**; g) Example of reading from keyboard**

(display "\nRead From Keyboard\n")

(define keyboardRead (read))

**; g.2) Example of reading from a file**

(display "\nRead From File\n")

(define inport (open-input-file "squared.txt"))

(read inport)

#|

**@@@ OUTPUT @@@**

Anonymous Function

25

Named Function

25

Function List

(square cube)

Compare List of Functions

(square cube)(cube square)(square cube)

#f

#t

#f

#t

Passing Functions

4096

68719476736

Returning a function as a result of another function

4

6

Read From Keyboard

Hello World

Read From File

'(define square (lambda (n) (\* n n)))

|#

**2. Write a Scheme function sigma that computes the standard deviation of any number of arguments.**

#lang racket

#|

Author: David Tran

File: sigma.rkt

Purpose: To implement a function sigma that computes the

standard deviation of any number of arguments (list).

|#

**;Function that computes the sum of the list.**

(define sum (lambda(list)

(if (null? list) 0 (+ (car list) (sum(cdr list))))))

**;Function computes the square of a number and returns it.**

(define square (lambda(x) (\* x x )))

**;Function that computes the sum of square of a list and returns it.**

(define sumSquared (lambda(list)

(if (null? list) 0 (+ (square(car list)) (sumSquared(cdr list))))))

**;Function that computes the standard deviation**

(define sigma (lambda(x . list)

(sqrt (- (/ (+ x (sumSquared list)) (+ (length list) 1)) (square(/ (+ x (sum list)) (+ (length list) 1)))))))

#|

**@@@OUTPUT@@@**

> (sigma 1 2 3 2 1)

0.7483314773547883

> (sigma 1 3 1 3 1 3)

1

> (sigma 1 3)

1

> (sigma 1)

0

> (sigma 1 2 3 4 5 6 7 8 9)

2.581988897471611

>

|#

**3.** **a) Write a recursive Scheme procedure line that prints n asterisks in a line.**

**b) Write a recursive Scheme procedure histogram that uses the procedure line, and prints a histogram for a list of integers.**

#lang racket

#|

Author: David Tran

File: asterisk.rkt

Purpose: To print \* depending on given parameters Ex. (line 5) will print \*\*\*\*\*

also to print \* given list as parameter Ex. (histogram '(1 2 3)) will print \* on first line and \*\* on second line

and \*\*\* on third line.

|#

**;displays \* depend on value of n passed to line**

(define line (lambda(n)

(if (< n 1) (display "")

(begin (display '\*) (line(- n 1))))))

**;displays \* depending on the value with in a list new line for every value**

(define histogram (lambda(list)

(if (null? list) (display "")

(begin (line(car list)) (display "\n") (histogram(cdr list))))))

#|

**@@@OUTPUT@@@**

> (line 10)

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

> (line 2)

\*\*

> (histogram '( 1 2 3))

\*

\*\*

\*\*\*

> (histogram '( 1 2 3 4 5))

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

>

|#

**4. Write a Scheme program for computing a maximum of function f(x) within the interval [x1, x2]. Use the trisection method, and find the coordinate of maximum xmax with accuracy of 6 significant decimal digits.**

#lang racket

#|

Author: David Tran

File: max.rkt

Purpose: Use the trisection method to find the maximum of

a function f(x)

Source: Dr. Jozo Dujmovic Programming Languages Reader pg. 174

|#

(define disp

(lambda (x n)

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

**;fmax function that finds the max given f, x1, x2**

(define fmax

(lambda(f x1 x2)

(cond

((< (- x2 x1) 1e-10)

(display "Maximum: f(")

(disp (/ (+ x1 x2) 2) 4)

(display ") = ")

(disp (f (/ (+ x1 x2) 2)) 4))

(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)))))))

#|

**@@@OUTPUT@@@**

> (fmax cos 1 2)

Maximum: f(1) = 0.5403

> (fmax sin 0 0)

Maximum: f(0) = 0

> (fmax sin 1 2)

Maximum: f(3927/2500) = 1.0

> (fmax cos 0 0)

Maximum: f(0) = 1

> (fmax tan 1 2)

Maximum: f(3927/2500) = 25253696228.7681

> (fmax tan 0 0)

Maximum: f(0) = 0

> (fmax tan 1 1)

Maximum: f(1) = 1.5574

>

|#

**5. Develop a program that computes the scalar product of two vectors. The program must not accept vectors having different size (in such a case print an error message).**

**a) Write the program in iterative style using the DO loop.**

**b) Write the program using recursion.**

#lang racket

#|

Author: David Tran

File: scalarProd.rkt

Purpose: To compute the scalar product of 2 vectors iteratively using a Do loop

and to compute the scalar product of 2 vectors recursively.

Source: Dr. Jozo Dujmovic Programming Languages Reader pg. 211

|#

**; a) Function for computing the scalar product of 2 vectors iteratively using Do loop.**

(define scalarProdIter

(lambda (v1 v2)

(if (= (vector-length v1) (vector-length v2)) ; checks if vectors length are equal

(begin (let ((s 0))

(do ((i 0 (+ i 1))) ; start of do loop from i = 0 to length of vector

((>= i (vector-length v1)) (display s))

(set! s (+ s (\* (vector-ref v1 i) (vector-ref v2 i)))))))

(display "Error: different size of vectors.")))) ; display error if vector lengths are not equal

**; b) Function for computing the scalar product of 2 vectors recursively**

(define scalarProdList

(lambda (list1 list2)

(cond ((null? (cdr list1)) (\* (car list1) (car list2))) ; condition check for empty lists

(else (+ (\* (car list1) (car list2)) ; add the head of each list

(scalarProdList (cdr list1) (cdr list2))))))) ; recurse by calling scalarProdList

(define scalarProdRec

(lambda (v1 v2)

(if (= (vector-length v1) (vector-length v2)) ;check if vector length are the same.

(scalarProdList (vector->list v1) (vector->list v2)) ; call to scalarProdList

(display "Error: different size of vectors.")))) ;display error if vector lengths are not equal

#|

**@@@OUTPUT@@@**

> (scalarProdIter '#(1 2 3) '#(2 1 1))

7

> (scalarProdRec '#(1 2 3) '#(2 1 1))

7

> (scalarProdIter '#(1 2 3) '#(2 1 1 1 1))

Error: different size of vectors.

> (scalarProdRec '#(1 2 3) '#(2 1 1 1 1))

Error: different size of vectors.

> (scalarProdIter '#(1 2 3 2 2) '#(2 1 1 1 1))

11

> (scalarProdRec '#(1 2 3 2 2) '#(2 1 1 1 1))

11

>

|#

**6. The files "matrix1.dat" and "matrix2.dat" are created using a text editor and contain two rectangular matrices.**

**a) Develop programs row and col that read a matrix from a file and display a specified row 3 or column.**

**b) Develop a program for matrix multiplication mmul that multiplies two matrices stored in specified input files, and creates and displays an output file containing the product.**

#lang racket

#|

Author: David Tran

File: matrix.rkt

Purpose: to create a program that reads in a matrix from a file and is able to display

a specific row or column. The program shall be able to take in 2 different file containing

matrices and perform matrix multiplication and outputing the results to another file.

Source: Dr. Jozo Dujmovic Programming Languages Reader pg. 218-219

|#

**;Read matrix from file**

(define (read-matrix filename)

(let\* ((inport (open-input-file filename))

(nrow (read inport))

(ncol (read inport))

(mat (make-vector nrow)))

(do ((i 0 (add1 i)))

((>= i nrow) (close-input-port inport) mat)

(let ((row (make-vector ncol)))

(do ((j 0 (add1 j)))

((>= j ncol) (vector-set! mat i row))

(vector-set! row j (read inport)))))))

**;Display components of the given vector**

(define (display-vector v)

(do ((i 0 (add1 i)))

((>= i (vector-length v)) (display ""))

(display (vector-ref v i)) (display " ")))

**;Return i-th row of a matrix**

(define (ro filename i)

(define mat (read-matrix filename))

(vector-ref mat i))

**;Display i-th row of a matrix**

(define (row filename i)

(display-vector (ro filename (- i 1))))

**;Return j-th col of a matrix**

(define (co filename j)

(define mat (read-matrix filename))

(define nrow (vector-length mat))

(define column (make-vector nrow))

(do ((i 0 (add1 i)))

((>= i nrow) column)

(vector-set! column i (vector-ref (vector-ref mat i) j))))

**;Display j-th col of a matrix**

(define (col filename j)

(display-vector (co filename (- j 1))))

**;Return dot product**

(define (dot-product v1 v2)

(do ((i 0 (add1 i)) (sum 0 (+ sum (\* (vector-ref v1 i)

(vector-ref v2 i)))))

((>= i (vector-length v1)) sum)))

**;Function performs matrix multiplication on 2 matrices in seperate files**

(define (mmul f1 f2 f3)

(define m1 (read-matrix f1))

(define m2 (read-matrix f2))

(define nrow (vector-length m1))

(define ncol (vector-length m2))

(define outport (open-output-file f3))

(display nrow outport) (display " " outport)

(display ncol outport) (newline outport)

(do ((i 0 (add1 i)))

((>= i nrow) (close-output-port outport) (display ""))

(let ((row (make-vector ncol)))

(do ((j 0 (add1 j)))

((>= j ncol) (display-vector row) (newline) (newline outport))

(vector-set! row j (dot-product (ro f1 i) (co f2 j)))

(display (vector-ref row j) outport) (display " " outport)))))

#|

**@@@OUTPUT@@@**

> (row "matrix1.dat" 2)

4 5 6

> (row "matrix2.dat" 2)

1 2 3

> (col "matrix1.dat" 2)

2 5

> (col "matrix2.dat" 2)

2 2 2

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

6 12 18

15 30 45

|#