Homework 4: Scheme

CSC 600-01 Programming Languages

Spring 2017

Emilio Quiambao

4 / 22 / 2017

1. First Class Objects

a) First class object expressed as an anonymous literal value (constant)

```
(( lambda(n) (* n n)) 2)
```

Output:

```
4
```

b) First class object stored in variables

```
(define (sqr x) (* x x))

(define (neg x) (* -1 x))

(sqr 5)

(neg 5)
```

Output:

```
25
-5
```

c) First class object stored in data structures

```
(define (sqr x)(* x x))

(define (neg x) (* -1 x))

(define listFunc (list sqr neg)) ;this is a list of 2 functions

listFunc
```

Output:

```
'(#<procedure:sqr> #<procedure:neg>)
```

d) First class object compared to other objects for equality

```
(define (sqr x)(* x x))

(define (neg x) (* -1 x))

(eqv? sqr neg)

(eqv? + +)

(equal? '(1 2 3) '(1 2 5))

(equal? '(1 2 3) '(1 2 3))
```

Output:

```
#f
#t
#f
#t
```

e) First class object passed as parameter to procedures/functions

```
(define (sqr x)(* x x))

(define (neg x) (* -1 x))

(neg (sqr (sqr 2)))
```

Output:

```
-16
```

f) First class object returned as a result from procedures/functions

```
(define (multiply x) (lambda(n)(* x n)))

((multiply 5) 4)
```

Output:

```
20
```

g) First class object read from the keyboard/file and display

READ EXAMPLE

```
(eval (read)) ;user will input
```

Output:

```
(* 5 6)
30
```

FILE EXAMPLE

```
(load “file.txt”) ;file contains (* 5 6)
```

Output:

```
30
```

DISPLAY EXAMPLE

```
(define answer (* 5 6))

(display answer)
```

Output:

```
30
```

2. Sigma function computing standard deviation

```
#lang racket
;Author: Emilio Quiambao
;Program Name: Standard Deviation
;Program Description: sigma function computing standard deviation
;Date: April 22, 2017
;Homework: 4 - Scheme
;Problem: 2

;summation of list
(define (sumList lst)
  (if (null? lst)
      0
      (+ (car lst)(sumList(cdr lst)))))

;summation squared
(define (sumListSq lst)
  (if (null? lst)
      0
      (+ (expt (car lst) 2)(sumListSq(cdr lst)))))

;mean of list
(define (meanList lst)
  (/ (sumList lst) (length lst)))

;mean squared
(define (meanListSq lst)
  (/ (sumListSq lst) (length lst)))

;standard deviation ie subtraction and square root
(define (sigmaList lst)
  (sqrt (- (meanListSq lst) (expt (meanList lst) 2))))
```

```
;sigma
(define (sigma . args) (sigmaList args))
```

Output:

```
> (sigma 1 2 3 2 1)
0.7483314773547883
> (sigma 1)
0
> (sigma 1 3 1 3 1 3)
1
> (sigma 1 3)
1
```

3. Recursive procedure printing asterisks in a line and histogram

```
#lang racket
;Author: Emilio Quiambao
;Program Name: Recursive Asterisk Printing and Histogram
;Program Description: recursive procedure that prints asterisks in a line & histogram
;Date: April 22, 2017
;Homework: 4 - Scheme
;Problem: 3



;(a) recursive asterik line procedure
(define (line x)
  (if (> x 0)
      (begin (display "*") (line (- x 1)))
      (newline)))
```

```
;(b) recursive histogram procedure
(define (histogram lst)
  (if (not (null? lst))
      (begin (line (car lst)) (histogram (cdr lst)))
      (void)))
```

Output:

```
> (line 5)
*****
> (line 10)
**********
> (histogram '(1 2 3 2 1))
*
**
***
**
*
```

4. Computing max within an interval using trisection method

```
#lang racket
;Author: Emilio Quiambao
;Program Name: Computing Maximum with Trisection Method
;Program Description: computes the max within an interval using trisection method
;Date: April 22, 2017
;Homework: 4 - Scheme
;Problem: 4

(define (displayRound x) (display (round x)))
(define fMax (lambda (func from to)
      (cond
            ((< (- to from) 1e-10)
                  (display "The maximum is ")
                  (displayRound (func (/ (+ from to) 2))))
      (else (let ((a1 (+ from (/ (- to from) 3)))
            (a2 (- to (/ (- to from) 3))))
            (if (< (func a1) (func a2))
                  (fMax func a1 to)
                  (fMax func from a2)))))))
```

Output:

```
> (fMax (lambda(x) (* x x))-1 1)
The maximum is 1
> (fMax (lambda(x) (add1 x))0 5)
The maximum is 6
```

5. Computing scalar product of two vectors

```
#lang racket
;Author: Emilio Quiambao
;Program Name: Scalar Product of Two Vectors
;Program Description: computes the scalar products of two vectors using two different
;                     methods (iteratively and recursively)
;Date: April 22, 2017
;Homework: 4 - Scheme
;Problem: 5

;(a) iterative DO loop
(define (scalProA v1 v2)
  (define sp 0)
  (if (equal? (vector-length v1) (vector-length v2))
      (begin
       (do ((i 0 (+ i 1)))
         ((> i (- (vector-length v1) 1)) sp)
         (set! sp (+ (* (vector-ref v1 i) (vector-ref v2 i)) sp))))
      (display "VECTOR SIZE ERROR")))

;(b) recursive
(define (scalProB v1 v2)
  (if (equal? (vector-length v1) (vector-length v2))
      (scalProList (vector->list v1) (vector->list v2))
      (display "VECTOR SIZE ERROR")))

(define (scalProList lst1 lst2)
  (if (> (length lst1) 0)
      (+ (* (car lst1) (car lst2))
         (scalProList (cdr lst1) (cdr lst2)))
      0))
```

Output:

```
> (scalProA '#(1 2 3) '#(2 1 1))
7
> (scalProB '#(1 2 3) '#(2 1 1))
7
> (scalProA '#(1 1 1) '#(2 1 3 4))
VECTOR SIZE ERROR
> (scalProB '#(1 2 4) '#(1 2))
VECTOR SIZE ERROR
```

6. Reading and displaying matrix from a file and multplication

```
#lang racket
;Author: Emilio Quiambao
;Program Name: Matrix Multiplication
;Program Description: reads the rows and columns of a matrix from a file and displays
;                     a specified row or column. Also does matrix multiplication.
;Date: April 22, 2017
;Homework: 4 - Scheme
;Problem: 6

;reading and displaying from a file
(define (row file r)
  (define inport (open-input-file file))
  (define numRows (read inport))
  (define numCols (read inport))

  (do ((i 1 (+ i 1)))
    ((> i (* (- r 1) numCols)) (display " "))
    (read inport))

  (do ((i 1 (+ i 1)))
    ((> i numCols) (newline))
    (display (read inport)) (display " ")))

(define (col file r)
  (define inport (open-input-file file))
  (define numRows (read inport))
  (define numCols (read inport))

  (do ((i 1 (+ i 1)))
    ((> i (* numRows numCols)) (newline))
    (if (= (modulo i numCols) r)
        (begin (display (read inport)) (display " "))
        (if (and (= r numCols) (= (modulo i numCols) 0))
            (begin (display (read inport)) (display " "))
            (read inport)))))

;multiplying matrices
(define (mmul f1 f2 f3)
      (define m1 (read-matrix f1))
      (define m2 (read-matrix f2))
      (define numRow (vector-length m1))
      (define numCol (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 numRow) (close-output-port outport) (display ""))
          (let ((row (make vector ncol)))
            do ((j 0 (add1 j)))
            ((>= j numCol) (display-vector row) (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" 1)
 1 2 3
> (row "matrix1.dat" 2)
 4 5 6
> (col "matrix1.dat" 1)
1 4
> (col "matrix1.dat" 3)
3 6
> (row "matrix2.dat" 1)
 1 2 3
> (row "matrix2.dat" 3)
 1 2 3
> (col "matrix2.dat" 1)
1 1 1
> (col "matrix2.dat" 2)
2 2 2
> (mmul "matrix1.dat" "matrix2.dat" "matrix3.dat")
6 12 18
15 30 45
```