Homework 6

scheme

CSC 600

28 June 2011

by Robin Pennock

Homework 6 CSC600
DrRacket used for development of all programs listed

```
1)      a) ((lambda(n m)(* m n))7 8)

        b)      > (define (afunct a b c)(+ a b c))
                > (afunct 1 2 3)
                6

        c)      > (define alist`(1 2 3 4 5))
                > (list? alist)
                #t

        d)      > (define bla 5)
                > (define bla2 5)
                > (= bla bla2)
                #t

        e)      (define (iffive n)
                        (cond[(= 5 n) "Yay!"]
                                [else "boo!"]))

                (define (printfive)5)
                ;
                ; test
                > (iffive(printfive))
                "Yay!"

        f)      (define (iffive n)
                        (cond[(= 5 n) "Yay!"]
                                [else printstuff]))

                (define printstuff "fail")
                ;
                ;test
                > (iffive 7)
                "fail"
                >
2)      ;problem 2
        ;sum is helper function for mean
        ;   this function simply calculates the sum
        ;   of all list items
        (define (sum list)
                (cond [ (null? list)0]
                        [else (+ (car list) (sum (cdr list)))]))

        ;helper function used by mean
        ;    calculates the number of items in a list
        (define (lengthof list)
                (cond [(empty? list) 0]
                         [else (+ 1 (lengthof (cdr list)))]))
        ;returns xbar from given list
        (define (mean list)
                (/ (sum list) (lengthof list)))
        ;returns sum of x^2/n
        (define (x2 list)
        (cond[(null? list)0]
                [else (/ (squaresum list) (lengthof list))]))
        ;squares and adds all values of list
        (define (squaresum list)
                (cond[(empty? list)0]
                        [else (+ (expt (car list) 2) (squaresum(cdr list)))]))

        ;calculate sigma using all previously defined helper functions
        (define (sigma . list)
                (cond [(null? list)0]
                        [else (sqrt(- (x2 list) (expt (mean list) 2)))]))

        ;;TEST RUN;;
        > (sigma 1 2 3 2 1)
        0.7483314773547883
        > (sigma 1 2 3 4 5)
        1.4142135623730951
        >
```

```
3)      a)      (define (line n)
                        (cond [(<= n 0) (newline)]
                                [else (begin
                                        (display `*)
                                (line(- n 1)))]))
        b)      (define (histogram list)
                        (cond [(null? list) (display "")]
                                [else (line(car list)) (histogram(cdr list))]))

                ;;TEST;;
                > (histogram `(1 2 3 2 1))
                *
                **
                ***
                **
                *
                > (histogram `( 1 2 32 4 23 4))
                *
                **
                ********************************
                ****
                ***********************
                ****
```

```
    >
```

4) this one was WAY harder than I thought!

```
;Problem 4

(define (find-max X Y funct)
;got multiply (x-y) by 1/3 from
;   http://www41.homepage.villanova.edu/robert.styer/trisecting%20segment/
;and
;
http://www.algebra.com/algebra/homework/Length-and-distance/Length-and-distance.faq.quest
on.345492.html

 ;also borrowing heavily from Dr Dujmovic's reader code!
 ;cant use define here since these are not identifiers
 ;just using general function instead of linear equation
 ;must use let*
 (let* ((trisect (/ (- Y X) 3.))
        (xtri (+ X trisect))
        (ytri (- Y trisect)))
;meat of the work
(cond [(> X Y)
          (display "ERROR: first value must be larger than second")]
          ;using abs library function
          ;if absolute value of functX - functY then divide (X + Y)/2
          ;basically keep running until reaching precision of 0.0000001
                   [(< (abs (- (funct X) (funct Y))) .0000001)
          ;bisection
                            (/ (+ Y X) 2.)]
          ;elseif (funct xtri < funct ytri) then findmax of X and ytri
          ;
          ;basically run again with a trisected Y value
          ;recursive call
                  [(> (funct xtri) (funct ytri))
                            (find-max X ytri funct)]
           ;Y must be bigger than xtri
          ;keep Y and run again with trisected x value
                  [else (find-max xtri Y funct)])))

;TEST RUN;
> (find-max -1 1 (lambda (X) (+ (- (* X X)) X)))
0.5000103753188725
> (find-max -1 3 (lambda (X) (- X (* X X X))))
0.5773135337279747
> (find-max -1 3 (lambda (X) (+ X (* X X X))))
2.9999999986059303
> (find-max -1 3 (lambda (X) (- X (* X X))))
0.499935459747847
```

```
>
```

5) a & b)

```
#lang scheme

;problem 5
;itterative scalar-product
;heavily borrowed from Dr Dujmovic's code
;except for the second line since DrRacket hates the '<>' operator

(define (scalar-product v1 v2)
  (cond [(not(equal? (vector-length v1) (vector-length v2)))
         (display "error: vectors not same length")]
        [(zero? (vector-length v1))(display "error: null vector")]
        [else (let((s 0))
                (do ((i 0 (add1 i)))
                  ((>= i (vector-length v1)) (display s))
                  (set! s (+ s (* (vector-ref v1 i) (vector-ref v2 i))))))]))

;very similar to Dr Dujmovic's splist dot-product code

(define (uselist list1 list2)
  (cond[(null? (cdr list1)) (* (car list1) (car list2))]
       [else (+ (* (car list1) (car list2))(uselist (cdr list1) (cdr list2))
                )]))

;same as scalar-product except the else statement converts the vectors
;   to lists for easier processing

(define (scalar-product-recursive v1 v2)
  (cond [(not(equal? (vector-length v1) (vector-length v2)))
         (display "error: vectors not same length")]
        [(zero? (vector-length v1))(display "error: null vector")]
        [else (uselist (vector->list v1) (vector->list v2))]))

;TEST;

> (scalar-product `#(1 2 3) `#(1 2 3))
14
> (scalar-product `#(1 2 3) `#(1 2))
error: vectors not same length
> (scalar-product `#() `#(1 2))
error: vectors not same length
> (scalar-product `#() `#())
error: null vector
> (scalar-product-recursive`#(1 2 3) `#(1 2))
error: vectors not same length
> (scalar-product-recursive`#() `#())
```

```
error: null vector
> (scalar-product-recursive`#(1 2 3) `#(1 2 3))
14
> (scalar-product-recursive`#(1 2 3) `#(3 2 1))
10
> (scalar-product`#(1 2 3) `#(3 2 1))
10
>
```

6 a)

```
#lang scheme

;Problem 6 a
;borrowing heavily from Dr Dujmovic's matrix handout
;this one is heavily commented to keep me losing track of variables
;incidently i love scheme but hate this () only syntax...
(define (row file rownum)
  ;definitions of local variables
  (define openfile (open-input-file file))
  ;total num of rows
  (define rowmax (read openfile))
  ;total num of cols
  (define colmax (read openfile))
  ;had to make this function internal to be able to use
  ;    local variables
  ;empty list at for storing entire row
  (define outrow '())

  (define (get-row row col)
    ;need to use if here 'cause multiple conds get confusing
    ;if (row == rownum) AND ( colmax > col)
    (begin (if [and (= row rownum) (> colmax col)]
                ;append read of openfile to list outrow
                (set! outrow (cons (read openfile) outrow))
                ;read char
                (read openfile))
           ;else
           (cond
             ;check row++ == rowmax
             [(= (+ 1 row) rowmax)
              ;and
              (if (> col colmax)
                  (display "")
                  ;get next char from next col
                  (get-row row (+ 1 col)))]
             [(= (+ 1 col) colmax)
              ;get-row on next row
              (get-row (+ 1 row) 0)]
             ;get-row on next column
             [else (get-row row (+ 1 col))])))
  (begin (set! rownum (- rownum 1))
  ;if specified row greater than rowmax
  (cond[ (< rowmax rownum)
      (begin (display "row #")
             (display rownum)
             (display " does not exist ")
             (close-input-port openfile))]
      [else (begin
             (get-row 0 0)
             (close-input-port openfile)
             ;since order will be revresed as these elements
             ;   were put into a list from top to bottom
             ;display reversed outrow
             (reverse outrow))])))
;end row

;begin column


(define (col file colnum)
  ;definitions of local variables
  (define openfile (open-input-file file))
  ;max num of rows in file
  (define rowmax (read openfile))
  ;max num of columns
  (define colmax (read openfile))
  ;empty list to be filled with chars by row
  (define column '())
  ;imbeded function so i have access to local vars
  (define (get-col row col)
    ;if (current col == colnum) then (read char from that column)
    (begin (if (= col colnum)
               (set! column (cons (read openfile) column))
               (read openfile))
           ;else
           (cond
             ;if (row++ == rowmax)
             [(= (+ 1 row) rowmax)
              ;reached end of current row, call next row for next column car
              (if (< colmax col )
                  (display "")
                  (get-col row (+ 1 col)))]
             [(= (+ 1 col) colmax)
              (get-col (+ 1 row) 0)]
             ;using row to get next line for get call to process
             [else (get-col row (+ 1 col))])))
  (begin (set! colnum (- 1 colnum))
  ;if specified col greater than colmax
  (if (> colnum colmax )
      ;isolted display and close
      (begin (display "col#")
             (display colnum)
             (display " Does not exist")
             (close-input-port openfile))
      ;display and close
      (begin (get-col 0 0)
             (close-input-port openfile)
             ;since order will be revresed as these elements
             ;   were put into a list from top to bottom
              ;display reversed column
              (reverse column)))))
```

```
;TEST RUN;

> fileloc1
"/home/rob/matrix1.dat"
> (row fileloc1 1)
(1 2 3)
> (row fileloc1 2)
(4 5 6)
> (row fileloc1 234234)
row #234233 does not exist
> (col fileloc1 1)
(1 4)
> (col fileloc1 234523452345)
col #234523452344 Does not exist
>
```

b) I ran out of time to do this one

My idea was to make a helper function that works similar to the scalar product function that processes vector multiplication 1 line at a time

Basically I was planning to use the (read-line) library function to go though and grab matrix data line by line until eof. And process using said helper function

This is as far as I got

```
;helper function
;used to see how long a row is
(define (lengthof list)
  (define bla 0)
(cond [(empty? list) 0]
      [else (+ 1 (lengthof (cdr list)))]))
```

```
        ;multiply row helper function
        ;this allows me to simply do matrix multiplication row by row
        ;fully working function
        (define (multrow lista listb)
        (define boundA (lengthof lista))
        (define boundB (lengthof listb))
        (define retvec(make-vector boundA))
        (cond[(> boundB boundA)(display "ERROR: lists of different size")]
                 [else (begin
                (set! boundB (- boundB 1))
                (do ((i 0 (+ i 1)))
                ((> i boundB))
                 (vector-set! retvec i (* (vector-ref veca i)(vector-ref vecb i))))
                 (define retlist(vector->list retvec))
                 (display retlist))]))

        (define (mmul file1 file2 file3)
        ;definitions of local variables
        (define openfile1 (open-input-file file1))
        (define openfile2 (open-input-file file2))
        (define openfile3 (open-input-file file3)
        (define row1 (read-line openfile1))
        (define row2 (read-ling openfile2))
        (define process-row)
                ;convert these rows to lists
                ;   ran into problems here with extra chars in read
                ;run lists through multrow
                ;output to file3
        ))

;TEST OF HELPER FUNCTIONS multrow;

> (multrow `(1 2 3 4) `(4 5 6 7))
(4 10 18 28)
>
```