Tarea 03 - Lenguajes de Programación

Programación Básica en Scheme

A01195067 Andres Cavazos

A00812572 Gustavo Ferrufino De La Fuente

1.- Implementar el predicado repetidos? que reciba cinco argumentos enteros y

que determine si alguno de ellos se repite

(define (repetidos? a b c d e)

( cond ( (or (equal? a b) (equal? a c) ( equal? a d) (equal? a e)) #t)

( (or (equal? c b) (equal? b d) (equal? b e)) #t)

( (or (equal? c d) (equal? c e)) #t)

(else #f)

)

)

2.- Implementar la función distintos que reciba 5 argumentos enteros y determine

cuantos valores distintos se dieron.

(define (distintos v1 v2 v3 v4 v5)

(unicos (list v1 v2 v3 v4 v5)))

(define (unicos list)

(if (null? (duplicados list))

0

(+ 1 (unicos (rest (duplicados list))))))

(define (duplicados v1)

(do ((a '() (if (member (car v1) a) a (cons (car v1) a)))

(v1 v1 (cdr v1)))

((null? v1) (reverse a))))

3.- Implementar la función recursiva pi que regrese el valor de pi calculado

mediante los n primeros términos de la serie que la define (mientras más

términos más se acercará al valor exacto).

(define (factorial n)

(if(<= n 1)

1

(\* n (factorial (- n 1)))

)

)

(define (power x y)

(cond ((= y 0) 1)

((> y 0) (\* (power x (- y 1)) x))

)

)

(define (Pi n)

(cond ((= n 0) 0)

((= n 1) 2)

(else ( + (\* 2 (/ (\* (power 2 (- n 1) ) (power (factorial (- n 1)) 2)) (factorial (+ (\* 2 (- n 1)) 1))))(Pi (- n 1))))

)

)

4.- Implementar la versión terminal de una función recursiva m2last que regresa

el n-ésimo elemento de la serie cuyos primeros 2 elementos son los números 1

y 2, y a partir del tercero se calcula como la multiplicación de los 2 elementos

anteriores. Los primero elementos de la serie son: 1, 2, 2, 4, 8, 32, 256, 8192,…

(define (func x y a n)

(cond[(< a n) (func y (\* x y) (+ 1 a) n)]

[(equal? a n) (\* x y)]))

(define (m2last n)

(cond[ (= n 1) 1]

[(= n 2) 2]

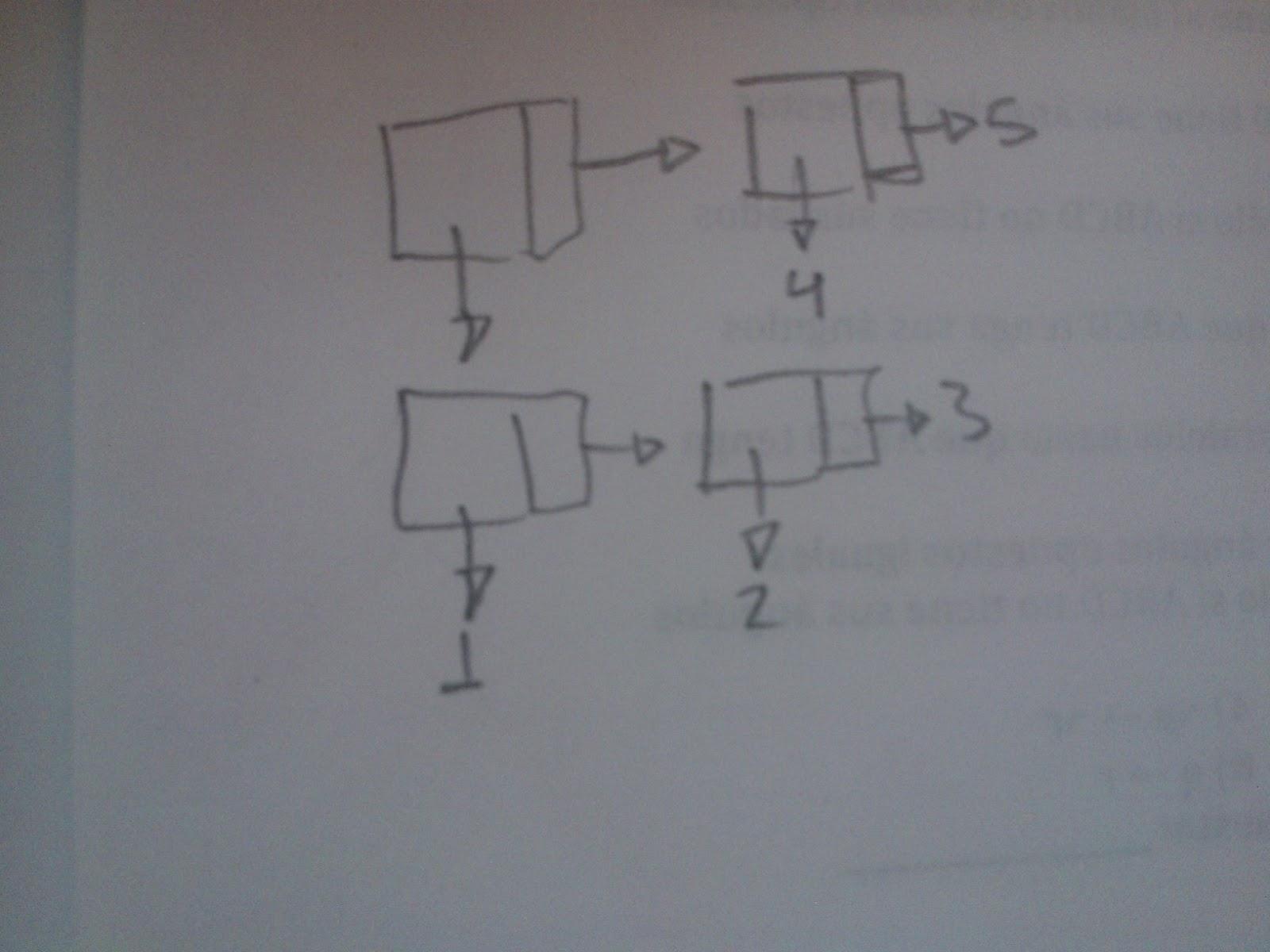
[else ( func 1 2 1 (- n 2))])

)

5.- Para cada uno de los siguientes incisos obtén las dos representaciones faltantes de las tres vistas en clase: representación visual, representación interna, y representación con el constructor cons (no se permiten otros constructores como list o append).

a.

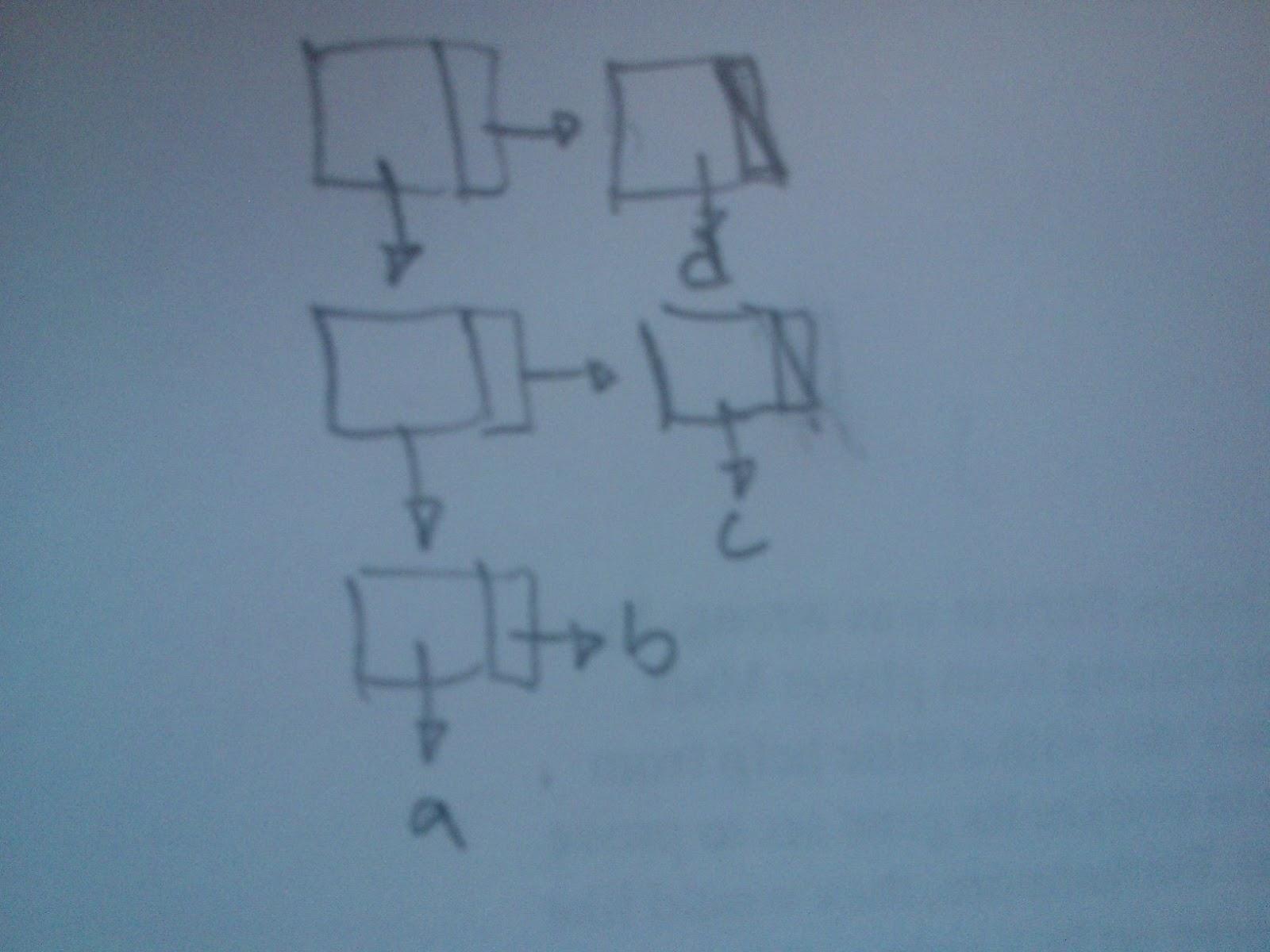
((1 2 . 3) 4 . 5)



(cons ( cons 1 (cons 2 3))(cons 4 5))

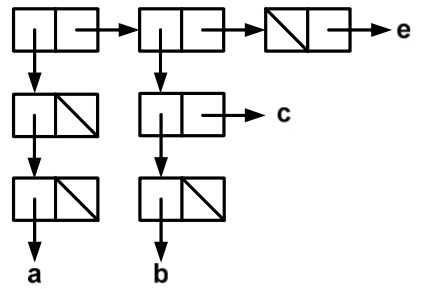
b.

(cons (cons (cons ‘a ‘b)(cons ‘c ‘())) (cons ‘d ‘()))



( ( (a.b).c) d )

c.



(cons (cons (cons 'a '()) '()) (cons( cons ( cons 'b '() )'c ) (cons '() 'e)))

( ( ( a ) ) ( ( b ) . c ) () . e )

6.- Implementar la función recursiva num->list que convierta su argumento dado

como un número entero no negativo en una lista de dígitos.

Probar con:

(define (num->list num)

(cond ((< num 10) (list num))

(else (append (num->list (floor (/ num 10))) (cons (remainder num 10) '())))

)

)

7.- Implementar la función recursiva repetidos que determine la cantidad de

elementos repetidos en 2 listas. Suponer que no hay elementos repetidos en

cada lista.

(define (repetidos l1 l2)

(cond [(and (null? l1) (null? l2)) 0]

((null? l1) (length l2))

((null? l2) (length l1))

(else (aux l1 l2 0))))

(define (aux l1 l2 cont)

(if (null? l1)

cont

(if (repetidoL2 (first l1) l2)

(aux (cdr l1) l2 (+ cont 1))

(aux (cdr l1) l2 cont))))

(define (repetidoL2 num l2)

(if (= ( length l2) 0)

#f

(if (equal? num (first l2))

#t

(repetidoL2 num (rest l2)))))

8.- Implementar la función recursiva mezcla que cree una lista ordenada de

enteros sin repetición a partir de dos listas ordenadas de enteros que pueden

tener valores repetidos.

; MERGE LISTS

(define (join l1 l2)

(if (null? l1) l2

(if (null? l2) l1

(cons (car l1) (cons (car l2) (join (cdr l1) (cdr l2)))))))

;ORDER ASCD

(define (ordAsc l1)

(cond ( (null? l1) '() )

( else (cons (smllt l1 (car l1))

(ordAsc (rem l1 (smllt l1 (car l1)))))

)

)

)

(define (rem l1 A)

(cond ( (null? l1) '() )

( ( = (car l1) A) (cdr l1))

(else (cons (car l1)(rem (cdr l1) A)))

)

)

(define (smllt l1 A)

(cond ( (null? l1) A)

( (< (car l1) A) (smllt (cdr l1)(car l1)))

(else (smllt (cdr l1) A ))

)

)

;REMOVE DUCPLICATED

(define (remove-duplicates list)

(if (or (null? list)

(null? (cdr list)))

list

(if (= (car list) (cadr list))

(remove-duplicates (cdr list))

(cons (car list) (remove-duplicates (cdr list))))))

;METODO PRINCIPAL

(define (mezcla l1 l2)

(remove-duplicates (ordAsc (join l1 l2))))