;;Author: Ebrain Mirambeau

;;Purpose: CSE 413 HW 1

;;SID: 0938789

;;problem 1

(define (! x)

(if (< x 1)

1

(\* x (! (- x 1)))))

(define (comb n k)

(/ (! n)

(\* (! k)(! (- n k)))))

;;tests

(display "problem 1: test cases\n")

(comb 0 0)

(comb 0 1)

(comb 12 3)

(comb 14 5)

;;problem 2

(define (zip lst1 lst2)

(zip-helper lst1 lst2 '()))

(define (zip-helper lst1 lst2 x)

(cond ((null? lst1) (append lst2 x))

((null? lst2) (append lst1 x))

(else (append (append (list (car lst1)) (append (list (car lst2)) x)) (zip-helper (cdr lst1) (cdr lst2) x)))))

;;tests

(display "problem 2: test cases\n")

(display "(zip '() '()) => ") (zip '() '())

(display "(zip '(1) '()) => ") (zip '(1) '())

(display "(zip '() '(1)) => ") (zip '() '(1))

(display "(zip '(1) '(2)) => ") (zip '(1) '(2))

(display "(zip '(1) '(2 3)) => ") (zip '(1) '(2))

(display "(zip '(1 3 5) '(2 4)) => ") (zip '(1) '(2))

(display "(zip '(2 4) '(1 3 5)) => ") (zip '(1) '(2))

(display "(zip '(1 2 3 8 5 3 4 4 1) '(4 5 6 7)) => ") (zip '(1 2 3 8 5 3 4 4 1) '(4 5 6 7))

;;problem 3

(define (unzip lst)

(unzip-helper '() '() lst))

(define (unzip-helper a b lst)

(if (null? lst) (cons a (list b))

(if (not (even? (length lst))) (unzip-helper (append a (list (car lst))) b (cdr lst))

(unzip-helper a (append b (list (car lst))) (cdr lst)))))

;;tests

(display "problem 3: test cases\n")

(display "(unzip '()) => ") (unzip '())

(display "(unzip '(1)) => ") (unzip '(1))

(display "(unzip '(1 2)) => ") (unzip '(1 2))

(display "(unzip '(1 2 3)) => ") (unzip '(1 2 3))

(display "(unzip '(1 2 3 4 5 6 89)) => ") (unzip '(1 2 3 4 5 6 89))

;;problem 4

(define (expand lst2)

(if (null? lst2)

(reverse lst2)

(if (and (list? (car lst2)) (= 2 (length (car lst2))))

(append (mult (car (car lst2)) (cadar lst2) '()) (expand (cdr lst2)))

(append (list (car lst2)) (expand (cdr lst2))))))

(define (mult num element lst)

(if (= num 0)

lst

(mult (- num 1) element (append (list element) lst))))

;;tests

(display "problem 4: test cases\n")

(expand '())

(expand '(1))

(expand '((3 a) 2))

(expand '(a (3 b) (3 a) b (2 c) (3 a)))

;;problem 5

;;a

(define (value lst)

(car lst))

(define (left lst)

(cadr lst))

(define (right lst)

(caddr lst))

;;tests

(display "problem 5 a: test cases\n")

(value '(1 () ()))

(left '(1 () ()))

(right '(1 () ()))

;;b

(define (size tree)

(if (null? tree) 0

(+ 1 (size (left tree)) (size (right tree)))))

;;test cases

(display "problem 5 b: test cases\n")

(display "(size '(1 (2 (3 (4 () ()) (5 (6 () ()) ()) ()) (7 () ())) (3 () ()))) => ")

(size '(1 (2 (3 (4 () ()) (5 (6 () ()) ()) ()) (7 () ())) (3 () ())))

;;c

(define (contains item tree)

(if (null? tree) #f

(if (equal? item (value tree)) #t

(or (contains item (left tree)) (contains item (right tree))))))

;;tests

(display "problem 5 c: test cases\n")

(contains 1 '(1 (2 () ()) (3 () ())))

(contains 2 '(1 (2 () ()) (3 () ())))

(contains 3 '(1 (2 () ()) (3 () ())))

(contains 4 '(1 (2 () ()) (3 () ())))

(contains 3 '(1 (2 (3 (4 () ()) (5 (6 () ()) ()) ()) (7 () ())) (3 () ())))

(contains 3 '(1 (2 (3 (4 () ()) (5 (6 () ()) ()) ()) (7 () ())) (3 () ())))

(contains 3 '(1 (2 (3 (4 () ()) (5 (6 () ()) ()) ()) (7 () ())) (3 () ())))

;;d

(define (leaves tree)

(leaves-helper '() tree))

(define (leaves-helper lst tree)

(cond ((null? tree) '())

((and (null? (left tree)) (null? (right tree))) (append (list (value tree)) lst))

(else (append (leaves-helper lst (left tree)) (leaves-helper lst (right tree))))))

;;tests

(display "problem 5 d: test cases\n")

(leaves '())

(leaves '(1 () ()))

(leaves '(1 (2 () ()) ()))

(leaves '(1 () (2 () ())))

(leaves '(1 (2 () ()) (3 () ())))

;;e

(define (isBST tree)

(cond ((null? tree) #f)

((null? (left tree)) (isBST (right tree)))

((null? (right tree)) (isBST (right tree)))

(else (and (< (value (left tree)) (value tree)) (> (value (right tree)) (value tree))))))

;;tests

(display "problem 5 e: test cases\n")

(isBST '())

(isBST '(1 () ()))

(isBST '(1 (1 () ()) (1 () ())))

(isBST '(1 (2 () ()) (3 () ())))

(isBST '(1 (3 () ()) (2 () ())))

(isBST '(2 (1 () ()) (3 () ())))

(isBST '(4 (2 (1 () ()) (3 () ())) (5 () ())))