THIRD QUIZ

You have 15 minutes from the start of class to complete this quiz. The usual instructions apply. Don’t reinvent the wheel—for full credit, use functions from an earlier problem where helpful in a later one. Good luck!

You may show lists in any of three ways: (list 73 15), '(73 15), or (cons 73 (cons 15 empty)).

Problem 1 (4 points)
Evaluate each of the following expressions. Use this definition independently for each of the four parts:
(define L (cons 'leaves (cons 'sweaters (cons 'rain (cons 'turkey empty)))))

(a) (first L)

(b) (rest L)

(c) (first (rest L))

(d) (rest (rest L))

(e) (cond
    ((empty? (rest (rest (rest L)))) 'Maine)
    ((empty? (rest (rest L))) 'Vermont)
    (else 'NewHampshire)))

Problem 2 (2 points)
What three lines does DrScheme produce when it evaluates this code?
(define election-years (cons 2000 (cons 2004 (cons 2008 empty))))
(rest election-years)
(first election-years)
(rest election-years)

Problem 3 (4 points)
Complete the definition of count-items-on-list below.

;; count-items-on-list: expression list -> number
;; Return the number of times the expression occurs on the list
(define count-items-on-list
    (lambda (item L)
        (cond
            (____________________ 0)
            ((equal? item ____________________) (+ 1 (count-items-on-list item (rest L))))
            (else (____________________ item __________________))))
Problem 4 (14 points)

A time is a structure (make-time ampm hour minute) where ampm is a symbol (‘am or ‘pm), hour is a number from 0 to 11, and minute is a number from 0 to 59:

(define-struct time (ampm hour minute))

(a) (3 points) The time in Pakistan is 12 hours different from the time in Irvine. That means we can convert Pakistan time to Irvine time (and vice versa) simply by changing ‘am to ‘pm or ‘pm to ‘am in the first field of the structure. Define the function switch-time that takes a time and returns a time, changed in that way.

;; switch-time: time -> time
;; Return time with first field changed (am->pm, pm->am)
(define switch-time
  (lambda (T)
    (make-time
      (cond ((symbol=? (time-ampm T) 'am) 'pm) (else 'am)) ; equal? is OK for symbol=?
            (time-hour T)
            (time-minute T))))

Scoring: 1 point for correctly returning a structure (i.e., make-time with three arguments); 1 point for correctly changing am/pm and pm/am; 1 point for all arguments being right.

(b) (5 points) Define the function switch-time-list that changes a list of Irvine times to Pakistan times.

;; switch-time-list: list-of-times -> list-of-times
;; For each time in the list, change am to pm and vice versa.
(define switch-time-list
  (lambda (L)
    (cond
      ((empty? L) empty)
      (else (cons (switch-time (first L)) (switch-time-list (rest L))))))

Scoring: 1 point for the empty case, 3 points for the non-consing case (e.g., 1 point for checking (first L) with time<? and cutoff, 1 point for an attempt at a recursive call with (rest L), and 1 point for getting it all right), 2 points for the consing case, with partial credit as warranted. Only –1 if they keep when they should have removed and vice versa; only –1/2 if they forget the “cutoff” argument in the recursive calls.

(c) (6 points) Suppose we already have defined the function time<? ("time less-than") that takes two times and returns true if the first time is earlier than the second time (and false otherwise).

Define the function remove-earlier-times that removes from a list of times all the items that are earlier than a specified cutoff time. Of course you should call time<? where necessary.

;; remove-earlier-times: list-of-times time -> list-of-times
;; Return a list containing all (and only) the times from the first input (a list of times) that are NOT earlier than the second input (the ”cutoff time")
(define remove-earlier-times
  (lambda (L cutoff)
    (cond
      ((empty? L) empty)
      ((time<? (first L) cutoff) (remove-earlier-times (rest L) cutoff))
      (else (cons (first L) (remove-earlier-times (rest L) cutoff))))))