CSCI 275

Lab 03: Structured Lists
Due Thursday, February 27 at 11:59 PM

Use map and apply wherever you can to write the following functions. If you can't see how to use map and apply I'll accept any other way to write them, but do make an effort to use higher-order functions. You can use the solution to any exercise as a helper function in subsequent exercises.

Part 1 – Vectors and Matrices as Lists

1. Write **(firsts llyst)** and **(rests llyst)**. For both of these llyst is a list of lists. (firsts llyst) returns a new list with the cars of each element of llyst; (rests llyst) returns a new list with the cdrs of each element of llyst. Examples:
   - (firsts '((a b c) (d e f) (g h i))) returns (a d g)
   - (rests '((a b c) (d e f) (g h i))) returns ((b c) (e f) (h i))

2. Write **(addvec vec1 vec2)**, where vec1 and vec2 are vectors (lists of numbers) with the same length. This returns a vector containing the sums of the corresponding elements of vec1 and vec2.
   - (addvec '(1 2 3) '(4 5 6)) returns (5 7 9)
   - (addvec '( ) '( )) returrns ( )

3. Write **(dot-product vec1 vec2)**, where again vec1 and vec2 are vectors with the same length. This returns the sum of the products of the corresponding elements of vec1 and vec2.
   - (dot-product '(1 2 3) '(4 5 6)) returns 32 (= 1*4 + 2*5 + 3*6)
   - (dot-product '( ) '( )) returns 0

4. We can represent a matrix by a list of vectors, where each vector has the same length and represents one row of the matrix. For examples, ((1 4 7) (2 5 8) (3 6 9)) represents

1	4	7
2	5	8
3	6	9

   Write **(dot-row vec mat)**, where the length of vec is the same as the length of each row of mat. This returns a vector containing the dot-product of vec with each row of mat.

- (dot-row '(1 2 3) '((1 4 7) (2 5 8) (3 6 9))) returns (30 36 42)
- (dot-row ‘(101) ‘((2 3 4) (1 1 1)) returns (6 2)

5. The transpose of a matrix interchanges its rows and columns i.e., the first row of mat is the first column of the transpose, the second row of mat becomes the second column of the transpose, and so forth. For example, the transpose of

1	4	7
2	5	8
3	6	9

is

1	2	3
4	5	6
7	8	9

Write function **(transpose mat)** which returns the transpose of matrix mat. Hint: use the functions firsts and rests from question (1). For example

- (transpose ‘((1 4 7) (2 5 8) (3 6 9)) returns ((1 2 3) (4 5 6) (7 8 9))
- (transpose '((1 2 3) (4 5 6))) returns ((1 4) (2 5) (3 6))

6. You have probably seen matrix multiplication before. The entry of the ith row and jth column of the product is the dot-product of row i of the first matrix and column j of the second matrix. For this to make sense the lengths of the rows of the first matrix must be the same as the lengths of the columns of the second. For example,

$$\begin{array}{|c|c|c|}\hline 1 & 0 & 1 \\ \hline 2 & 1 & 1 \\ \hline \end{array} \quad X \quad \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 1 & 0 \\ \hline 1 & 1 \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 4 & 5 \\ \hline \end{array}$$

Write function **(matmult mat1 mat2)** that returns the product of matrices mat1 and mat2. Hint: use a helper function that works with mat1 and the transpose of mat2. This helper function makes a lot of use of dot-row.
For example

- (matmult '((1 0 1) (2 1 1)) '((1 2) (1 0) (1 1))) returns ((2 3) (4 5))

Part 2 – Functions that operate on non-flat lists

7. Write **(flatten L)** that takes a list L and creates a flat list wth the same elements.
   - (flatten   '(x y z z y)) returns(x y z z y)
   - (flatten   '(a (x (y)) (((y)) y z)) returns (a x y y y z)

8. Consider a list, not necessarily flat, of numbers, such as ((1 (2)) (((4))) 5). Use map and apply to write a procedure **(sum L)** that sums all of the numbers in such a list. For example,

   - (sum ‘((1 (2)) (((4))) 5)) returns 12

9. Again consider a general list of numbers and a function f of one argument. Use map and apply to write a procedure **(map-to f L)** that builds a new list with the same structure as L, only f is applied to each of the elements of L to get the values in the new list. For example,

   - (map-to add1 '(3 (4 5))) returns (4 (5 6))

10. Write the procedure **(element-of? a L)** that returns #t if atom a is an element of (not-necessarily flat) list L, and #f otherwise

   - (element-of? 3 ‘(2 1 (4 2 (5 3) 1))) returns #t
   - (element-of? ‘x ‘(a (b c (d)) e f)) returns #f