Lisp: Question 1

Write a recursive lisp function that takes a list as an argument and returns the number of atoms on any level of the list. For instance, list (A B (C D E) ()) contains six atoms (A, B, C, D, E, and NIL).

(defun count-atoms (x)
  (cond ((null x) 0)
    ;; No more children.
    ((not (listp x)) 1)
    ;; Terminal node.
    (t (+ (if (atom (first x)) 1 (count-atoms (first x)))
      ;; Break the problem down into two subproblems.
      (count-atoms (rest x))))))
Question 1: count-atoms

[2]> (count-atoms '(A B C))
3
[3]> (count-atoms '(A B C nil))
4
[4]> (count-atoms '(A B C (nil (A B))))
6
[5]> (count-atoms '(A B C (nil (A B ()))))
7
[6]> (count-atoms '(()))
1
[7]> (count-atoms '(()))
1
[8]> (count-atoms '(() A B C))
4
[9]>
Question 2: last5

Write a lisp function last5 that takes a list A as its argument and returns a list B consisting of the last five elements of A. You are not allowed to use the built-in function last.

(last5 '(A B C)) should return (A B C)
(last5 '(A B C D E F G H)) should return (D E F G H)

(defun last5 (x)
  (cond ((null (rest (rest (rest (rest (rest x))))))) x)
    (t (last5 (rest x)last5))))

[3]> (last5 '(1 2 3 ))
(1 2 3)
[4]> (last5 '(1 2 3 4 5 6 7 8 9 10 11))
(7 8 9 10 11)
[5]> (last5 nil)
NIL
Question 3: flip

Write a recursive function flip that takes a binary tree as input and returns a binary tree that it is its mirror image. You can represent binary trees as nested structures:

Nested (recursive) representation: (\texttt{<root>} (\texttt{<left subtree>}) (\texttt{<right subtree>}))

Examples:

\begin{align*}
(\text{flip } '(1 2 3)) & \quad \text{should return } (1 3 2) \\
(\text{flip } '(1 (2 3 4) ()))) & \quad \text{should return } (1 () (2 4 3)) \\
(\text{flip } '(1 (2 (3 4 5) (10 11 12)) (6 () (7 () 8)))) & \quad \text{should return } (1 (6 (7 8 ()) ()) (2 (10 12 11) (3 5 4)))
\end{align*}
Question 3: flip

(defun flip (x)
    (list (first x)
        (if (atom (third x)) (third x)
            (flip (third x)))
        (if (atom (second x)) (second x)
            (flip (second x))))

[14]> (flip '(1 2 3))
(1 3 2)
[15]> (flip '(1 (2 3 4) ()
(1 NIL (2 4 3))
[16]> (flip '(1 (2 (3 4 5) (10 11 12)) (6 () (7 () 8)))
(1 (6 (7 8 NIL) NIL) (2 (10 12 11) (3 5 4)))
[17]>
Simple Lisp Functions

a) Write a lisp function `funny_first` that takes a list of flat lists and returns a new list composed of the first elements of the original flat lists.

b) Write a lisp function `funny_last` that takes a list of flat lists as its argument and returns a new list composed of the last elements of the original flat lists.

c) Write a lisp function `funny_len` that takes a list of flat lists as its argument and returns the sum of the lengths of the nested lists.

d) Write a lisp function `funny_sum` that takes a list of flat lists of numbers and returns the sum of the elements of the nested lists.

(funny_first '((A B) (C) (D E) (F G H))) should return (A C D F)
(funny_last '((A B) (C) (D E) (F G H))) should return (B C E H)
(funny_len '((A B) (C) (D E) (F G H))) should return 8
(funny_sum '((1 2) (3) (4 5) (10 20 30))) should return 75
(defun funny_first (x)
  (mapcar #'(lambda (y) (first y)) x))

(defun funny_last (x)
  (mapcar #'(lambda (y) (first (last y))) x))

(defun funny_len (x)
  (apply #'+ (mapcar #'(lambda (y) (length y)) x)))

(defun funny_sum (x)
  (apply #'+ (mapcar #'(lambda (y) (apply #'+ y)) x)))

[30]> (funny_first '((A B) (C) (D E) (F G H)))
(A C D F)

[31]> (funny_last '((A B) (C) (D E) (F G H)))
(B C E H)

[32]> (funny_len '((A B) (C) (D E) (F G H)))
8

[33]> (funny_sum '((1 2) (3) (4 5) (10 20 30)))
75
Question: ListNonNumbers

Write a lisp function that takes a flat list as an argument and returns a list whose elements are those elements of the original list that are not numbers.

(defun ListNonNumbers (x)
  (mapcan #'(lambda (y) (if (numberp y) nil (list y))) x))

[40]> (ListNonNumbers '(A B C D 3 5 6))
(A B C D)

[41]> (ListNonNumbers '(A B C D 3 5 6 (2 3 4)))
(A B C D (2 3 4))

[42]> (ListNonNumbers '(A B C D 3 5 6 (2 3 4) nil))
(A B C D (2 3 4) NIL)

[43]> (ListNonNumbers nil)
NIL
Question: AddNumbers

Write a lisp function that takes a flat list as an argument and returns a sum of the numbers in the original list. Your function should not add the non-number elements of the original list.

(defun AddNumbers (x)
  (apply #'+ (mapcar #'(lambda (y) (if (numberp y) y 0)) x)))

[45]> (AddNumbers ' (A B C D 3 5 6 (2 3 4) nil))
14
[46]> (AddNumbers ' (A B C D 3 5 6))
14
[47]> (AddNumbers ' (1 2 3 4 5 6))
21
[48]> (AddNumbers ' (A B C D nil (2 3 4)))
0
Question: d_shuffle

Write a lisp function \textit{d_shuffle} that takes a list of 32 different symbols and returns a list in which the first 16 original symbols are interleaved with the second 16 original symbols, i.e. list \((s_1 \ s_2 \ s_3 \ s_4 \ldots s_{29} \ s_{30} \ s_{31} \ s_{32})\) becomes \((s_1 \ s_{17} \ s_2 \ s_{18} \ldots s_{15} \ s_{31} \ s_{16} \ s_{32})\).

\begin{verbatim}
(defun d_shuffle (l)
  (do ((newl nil) (i 15 (- i 1)))
      ((< i 0) newl)
    (setf newl (cons (nth i l)
                      (cons (nth (+ i 16) l) newl)))))
\end{verbatim}

\begin{verbatim}
[53]> (d_shuffle '(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32))
(1 17 2 18 3 19 4 20 5 21 6 22 7 23 8 24 9 25 10 26 11 27 12 28 13 29 14 30 15 31 16 32)
[54]>
\end{verbatim}
Water-Jug Puzzles

In the water-jug puzzle we are given a 4-liter jug, and a 7-liter jug. Initially, both jugs are empty. Either jug can be filled with water from a tap, and we can discard water from either jug down a drain. Water may be poured from one jug into the other. There is no additional measuring device. We want to find a set of operations that will leave precisely $x$ liters of water in either one of the jugs.

i. Set up a state-space search formulation of the water jug puzzle:
   a) Given the initial iconic state description as a data structure.
   b) Give a goal condition on states as some test on data structures.
   c) Name the operators on states and give precise descriptions of what each operator does to a state description.

ii. Find whether the goals $x = \{1, 2, 3, 4, 5, 6, 7\}$ can be accomplished in 8 or fewer steps.

Hint: Use breadth-first search.
Water-Jug Puzzle

a) \((A \ B)\)  // A is the amount in the 4-liter jug  
   // B in the 7-liter jug
b) \((A == x)\) or \((B == x)\)

c) \(FA: (4 \ B),\)
    \(FB: (A \ 7)\)
    \(EA: (0 \ B),\)
    \(EB: (A \ 0)\)

\[\text{PAB: if } ((A+B) \leq 7) \text{ then } (0 \ A+B) \]
\[\text{else } (A+B-7 \ 7)\]
\[\text{PBA: if } ((A+B) \leq 4) \text{ then } (A+B \ 0) \]
\[\text{else } (4 \ A+B-4)\]
Water-Jug Puzzle Solution