CSC 351L – Comparative Programming Languages Lab
Project 7 – Lisp Control Structures
Due: April 22, 2021 at 11:59 PM

Purpose:

The purpose of this project is to give you more experience with recursion and to introduce you to Lisp control structures.

Steps:

Note: You may use functions created in the early steps to help you solve the problems in the later steps. Also read each step carefully. The word "recursive" appears on purpose in the descriptions.

1) Define a Lisp function called `headList` that when given a list of sublists will return a list with just the first element of each sublist. So, `(headList '((a b) (c d e) (f g) (h) (i j k)))` would return `(a c f h i)`. You must use the `mapcar` function for this answer.
2) Define a recursive Lisp function called `lastElement` that will return the last element of a list. So, `(lastElement '(a b c d))` would return `d`. You can assume the list has at least one element.
3) Define a recursive Lisp function called `allButLast` that will return the list without the last element. So, `(allButLast '(a b c d))` would return `(a b c)`. You can assume the list has at least one element. You should do this recursively and not use the remove function.
4) Define a Lisp function called `tailList` that when given a list of sublists will return a list with just the last element of each sublist. So, `(tailList '((a b) (c d e) (f g) (h) (i j k)))` would return the list `(b e g h k)`. You must use the `mapcar` function for this answer.
5) Define a recursive Lisp function called `removeApair` that takes a list of atoms and returns a list with the first pair of adjacent matching symbols removed. For example, the call `(removeApair '(a b c c d d b e a))` would return the list `(a b d d b e a)`.
6) Define a recursive Lisp function called `removePairs` that when given a list of atoms will return a list that has no two adjacent elements that are the same. For example, the call `(removePairs '(a b c c d d b e a))` would return the list `(a e a)`. Notice that this has to be done recursively because removing adjacent pairs might create a new set of adjacent pairs that need to be removed.
7) Define a Lisp function called `rotate` that will rotate the elements of a list by one position putting the last element first. This function does not need loops, assignments, helper functions, or the mapcar

function. This can be done with one statement. For example, the call `(rotate '(a (c d) e))` would return the list `(e a (c d))`. You can assume the list has at least one element.

8) Define a Lisp function called `spin` that will accept a list and a number and rotate the last element of the list to the front of the list the number of times indicated. For example, the call `(spin '(a b c d e f g) 3)` would return the list `(e f g a b c d)`. You can assume the list has at least one element. This is the only part of this project that can use a do loop and the `setf` function.

Deliverables:

When you complete the program, you will prepare two things: a project report and a zip file containing all of your code.

The project report must use the format given in the sample file on D2L. This report can be prepared as a text file, MS Word document, or PDF. The project report should not be included in the zip file.

The zip file should include all lisp files. You can create the zip file in one of two ways – using zip on brahe or on your own computer.

The two items must be uploaded to the Project 7 drop box on D2L. No other form of submission will be accepted.