Assignment #2 – MORE LISP
CSCI E-220 Artificial Intelligence
Due Thursday September 15, 2011

Submit, both electronically and on paper, both your code for the assignment and a runtime session showing the functions in use on an appropriate range of test data.

When you have all your files in one place you may email them to the TA, or submit them online using the submit function. The syntax for the submit function is:

    submit lib220 2 ~/asst2

assuming that ~/asst2 is the directory you've created to store the files for this assignment.

Problem 1.
(a) Define a recursive function ONLY-ATOMS that takes a list as its single argument, and returns T if the list contains only atomic values. It should return NIL if the list contains any non-atomic values (e.g. sublists).

    (b) Then write the same function using a looping (iterative) construct (that is, include one of the functions LOOP, DO, DOLIST, or DOTIMES in your solution). Give your second function a different name than the first!

Problem 2.
(a) Write a recursive function called POWER. POWER should take two arguments, \textit{x} and \textit{y}, and should return \textit{x}^{\textit{y}}. You may not use the built-in power function or trigonometric functions in your answer. You may use addition, subtraction, multiplication, or division as needed. Remember, your function must be recursive. If \textit{x} is not a number or \textit{y} is not a non-negative integer, POWER should return NIL.

    (b) Then write the same function using a looping (iterative) construct. Give your second function a different name.

Problem 3. Write a recursive function SQUASH to squash a list flat; e.g. the call

    (SQUASH ' (A ((B C) D (E)) F (((G))) ))

should return (A B C D E F G). For this assignment, NIL should be treated as a list rather than as an atom. (There should be no occurrences of the atom “NIL” in any output from your function).

Problem 4. Write a function COUNT which takes an atom and a list, and returns the number of times the atom occurs in the list, at any level. For example, the call

    (COUNT 'A ' (A B C A (A (A B))))

should return 4. Try not to use any local or global variables other than the formal arguments to the function, but do consider using a helper (auxiliary) function. Do not use the built-in functions COUNT, COUNT-IF, or COUNT-IF-NOT in your solution. Hint: COUNT can call SQUASH.
**Problem 5.** Next week you will be writing programs to search through a map for the best route from point A to point B. In preparation for that assignment, here is a step-by-step conceptual plan for writing a search program that looks for a particular atom in a list and returns the path to it.

Data structure: Leaves and nodes of the tree are atoms. The root of a tree is the CAR of a list. The other elements of the list are the branches. So, to use the tree shown at the right, start with the command

\[
\text{(SETQ TREE '(S (A (C)) (B (D (F)) (E (G)))))}
\]

Statement of problem: Write a function (MY-FIND GOAL TREE) to find a path to a given atom, returning the complete list of atoms encountered along the way. (Note: we call the function MY-FIND because a built-in function named FIND already exists.)

Usage Example 1:
- Input: (MY-FIND ‘F TREE)
- Expected output: (S B D F)

Usage Example 2:
- Input: (MY-FIND ‘S TREE)
- Expected output: (S)

Usage Example 3:
- Input: (MY-FIND ‘X TREE)
- Expected output: NIL

Recursive algorithm in English:
- **CASE 1 (base):** If the root of the tree we’re examining is the goal, then we return a list consisting of the goal atom and we’re done.
- **CASE 2 (failure):** If the goal atom is not contained anywhere in TREE, then we fail with NIL. (Note: you can call SQUASH as part of testing test this hypothesis).
- **CASE 3 (recursion):** At this point, we know that the root of the tree is part of the path we want to follow, but it’s not the final node, and we don’t know which way to go. So let’s iterate through the remaining branches of the tree, calling FIND recursively with each branch, and exiting the iteration if we find a solution.

Pseudocode for (MY-FIND GOAL TREE):

**BASE CASE:**
- IF GOAL=(CAR TREE), return (LIST (CAR TREE))

**FAILURE CASE:**
- IF GOAL is not a MEMBER of (SQUASH TREE), return NIL

**RECURSIVE CASE** (any case other than above two):
- IF T, return CONS of (CAR TREE) and BRANCH-OUT GOAL (CDR TREE)

**BRANCH-OUT** is a new function which takes non-rooted trees, e.g. ((A (C)) (B (D (F)) (E (G))))

Pseudocode in English for (BRANCH-OUT GOAL BRANCHES):
- DOLIST across the BRANCHES, with index L equal to each branch in turn, calling (MY-FIND GOAL L). Return immediately (don’t keep iterating) if FIND returns non-NIL.