FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE - CS161
Fall 2013

Programming Assignment 1 - Due 11:59pm October 10, 2013

Homework Guidelines:

- Submit your commented LISP program in a file **named hw1.lsp via Courseweb.**

- In a separate file **named test.txt**, submit **via Courseweb** a sample execution showing the values your programs return for the test cases indicated below.

- Your programs should be written in good style. In LISP, a comment is any characters following a semicolon (;) on a line. Provide an overall comment explaining your solutions. Furthermore, every function should have a header comment explaining precisely what its arguments are, and what value it returns in terms of its arguments. In addition, you should use meaningful variable names.

- The physical layout of the code on the page is very important for making LISP programs readable. Make sure that you use blank lines between functions and indent properly. Programming style will be a major consideration in grading the assignment.

- You are restricted to using the following functions, predicates, and operators introduced in class: quote ['], car, cdr [cadadr, etc.], first, second [third, etc.], rest, cons, list, append, length, numberp, stringp, listp, atom, symbolp, oddp, evenp, null, not, and, or, cond, equal, defun, let, let*, =, +, -, *, /. Note: you are <u>not</u> permitted to use setq or last.

- You may assume that all input to your functions are legal; i.e. you do not need to validate inputs.

- Do not write any additional helper functions for your code unless this is explicitly allowed. Test functions are OK.

- Your function declarations should look exactly as specified in this assignment. Make sure the functions are spelled correctly, take the correct number of arguments, and those arguments are in the correct order.

- Even if you are not able to implement working versions of these functions, please include a correct skeleton of each. Some of these assignments are auto graded and having missing functions is problematic.

1. Write a single LISP function, called SUB-LIST, that takes a list L and two non-negative integers START and LEN, and returns the sub-list of L starting at position START and having length LEN. Assume that the first element of L has position 0.

For example,

(SUB-LIST ‘(a b c d) 0 3) returns (a b c);
(SUB-LIST ‘(a b c d) 3 1) returns (d);
(SUB-LIST ‘(a b c d) 2 0) returns NIL.

- if the first parameter is greater than or equal to the length of the list, return NIL.
- if the second parameter is too large that it goes beyond the elements of the list, just return whatever elements it covers.

2. Write a single LISP function, called SPLIT-LIST, that takes a list L, and returns a list of two lists L1 and L2, in that order, such that

- L is the result of appending L1 and L2
- Length of L2 minus length of L1 is 0 or 1.

For example,

(SPLIT-LIST ‘(a b c d)) returns ((a b) (c d));
(SPLIT-LIST ‘(a b c d e)) returns ((a b) (c d e));   NOTE: ((a b c) (d e)) is incorrect;
(SPLIT-LIST ‘(a b c d e f)) returns ((a b c) (d e f)).

You can call the function SUB-LIST from SPLIT-LIST.

3. A binary tree can be represented as follows:

- if the tree contains a single node N, it can be represented by atom N
- if the tree has more than one node and is rooted at N, then it can be represented by a list (L R) where L represents the left subtree of N and R represents the right subtree of N

Write a single LISP function, called LIST2BTREE, that takes a non-empty list of atoms LEAVES, and returns a binary tree such that

- The tree leaves are the elements of LEAVES
- For any internal (non-leaf) node in the tree, the number of leaves in its right branch minus the number of leaves in its left branch is 0 or 1.

For example,

(LIST2BTREE '(1)) returns 1;
(LIST2BTREE '(1 2)) returns (1 2);
(LIST2BTREE '(1 2 3)) returns (1 (2 3)) ;
(LIST2BTREE '(1 2 3 4)) returns ((1 2) (3 4));

(LIST2BTREE '(1 2 3 4 5 6 7)) returns ((1 (2 3)) ((4 5) (6 7)));
(LIST2BTREE '(1 2 3 4 5 6 7 8)) returns (((1 2) (3 4)) ((5 6) (7 8))).

You can call the function SPLIT-LIST from LIST2BTREE.

4. Write a single LISP function, called BTREE2LIST, that takes a binary tree TREE, and returns a list of atoms. (Assume TREE follows the constraints we defined in problem 3)

- As the input is a binary tree, each node has at most 2 children.
- This function is the inverse of LIST2BTREE. That is, (BTREE2LIST (LIST2BTREE X)) = X for all lists of atoms X.

For example,

(BTREE2LIST 1) returns (1);
(BTREE2LIST '(1 2)) returns (1 2);
(BTREE2LIST '(1 (2 3))) returns (1 2 3) ;
(BTREE2LIST '((1 2) (3 4))) returns (1 2 3 4);
(BTREE2LIST '((1 (2 3)) ((4 5) (6 7)))) returns (1 2 3 4 5 6 7);
(BTREE2LIST '(((1 2) (3 4)) ((5 6) (7 8)))) returns (1 2 3 4 5 6 7 8).