**FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE - CS161**
**Fall 2018**

*Programming Assignment 1 - Due 11:55pm Tuesday, October 9*

## Homework Guidelines:

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

- Your programs will be evaluated under CLISP interpreter. In order to get any scores, you need to make sure that the following LISP command does not produce any errors in CLISP interpreter:

  > (load "hw1.lsp")

- 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.

- 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, if, equal, defun, let, let*, =, <, >, +, -, *, /. Note: you are **not** permitted to use setq.

- 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.

- 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.

An ordered tree is either a number n or a list (L m R), where

- L and R are ordered trees;
- m is a number;
- all numbers appearing in L are smaller than m;
- all numbers appearing in R are larger than m.

Some examples of ordered trees: 3, (1 2 3), ((1 2 3) 7 8), ((1 2 3) 5 (6 8 (9 10 (11 12 13)))).

**1.** Write a single Boolean LISP function, called TREE-CONTAINS, which takes two arguments N and TREE, and checks whether number N appears in the ordered tree TREE.

For example,

(TREE-CONTAINS 3 '((1 2 3) 7 8)) returns T
(TREE-CONTAINS 4 '((1 2 3) 7 8)) returns NIL

**2.** Write a single LISP function, called TREE-MIN, which takes one argument TREE, and returns the minimum number appearing in the ordered tree TREE.

For example,

(TREE-MIN '((1 2 3) 7 8)) returns 1

**3.** Write a single LISP function, called TREE-ORDER, which takes one argument TREE, and returns an pre-ordered list of the numbers appearing in the ordered tree TREE.

For example,

(TREE-ORDER 3) returns (3)
(TREE-ORDER '((1 2 3) 7 8)) returns (7 2 1 3 8)

**4.** 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) return s NIL

**5**. 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 L1 minus length of L2 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.

A binary tree is one in which each node has 0 or 2 children. A node that has 0 child is called a leaf node. A node that has 2 children is called an internal node. A binary tree can be represented as follows:

- A leaf node N is represented by atom N;
- An internal node N is represented by a list (L R), where L represents the left child of N and R represents the right child of N.

**6.** Write a single LISP function, called BTREE-HEIGHT, which takes a binary tree TREE, and returns the height of TREE. Note that the height of a binary tree is defined as the length of the longest path from the root node to the farthest leaf node.

For example,

(BTREE-HEIGHT 1) returns 0
(BTREE-HEIGHT '(1 2)) returns 1
(BTREE-HEIGHT '(1 (2 3))) returns 2
(BTREE-HEIGHT '((1 2) (3 4))) returns 2
(BTREE-HEIGHT '((1 (2 3)) ((4 5) (6 7)))) returns 3
(BTREE-HEIGHT '(((1 2) (3 4)) ((5 6) (7 8)))) returns 3

**7**. 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 left branch minus the number of leaves in its right 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.

**8**. Write a single LISP function, called BTREE2LIST, that takes a binary tree TREE as input, and returns a list of atoms (assume TREE follows the constraints we defined earlier).

- 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)

**9**. Write a single Boolean LISP function, called IS-SAME, that takes two LISP expressions E1 and E2 whose atoms are all numbers, and checks whether the expressions are identical. In this question, you can only use '=' to test equality (you cannot use 'equal'). Recall that a LISP expression is either an atom or a list of LISP expressions.

For example,

(IS-SAME '((1 2 3) 7 8) '((1 2 3) 7 8)) returns T
(IS-SAME '(1 2 3 7 8) '((1 2 3) 7 8)) returns NIL