Homework 8: LISP and Functional Programming

This homework is due Thursday, April 6. Each weekday that the homework is late you lose 10% of the possible points.

Except for the last problem, the LISP code, clearly documented and in order of the problems below, should be mailed to `matthews` on Zaphod[1]. Your solutions will be graded on their correctness, style, and documentation. Please test your functions, but do not send us the test data or test results.

Please program in an applicative style, using lists. That is, for this homework you may *not* use the imperative constructs of LISP (such as `set`, `setq`, `setf`, `rplaca`, `rplacd`, `nconc`, or `delete`), arrays, or the "program feature" (`prog`, `prog1`, `prog2`, or `progn`). You may assume that argument lists are not circular.

1. The following are some warm-up exercises in LISP.

   (a) (2 points) Write a function `seventh` that returns the seventh element of a list (which can be assumed to have at least seven elements).

   (b) (5 points) Write a function `copy-list` that returns a copy of the top level of the list structure in its argument. If the argument is `nil` it is returned, otherwise the result is a list consisting of argument's car and a copy-list of the argument's cdr.

   (c) (5 points) Write a function `copy-tree` that returns a copy of a tree of cons cells. If the argument is an atom it is returned; otherwise the result is a new cons consisting of a copy-tree of the car and cdr of the cell.

   (d) (5 points) Write a function `gcd` that computes the greatest common divisor of two positive integers. (The greatest common divisor of two positive integers m and n is the largest integer that evenly divides m and n.)

   (e) (5 points) Write a function `union` that takes two argument lists u and v (neither of which has any duplicate elements), and returns a list that contains just the elements of both u and v, without duplicates. For example `(union '(a b) '(a c)` should return a list containing `a`, `b`, and `c`, in any order. Use `eql` to test whether two elements are the same.

2. (10 points) Write a function `postorder` that takes a binary tree as an argument and returns a list consisting of the labels of the nodes in the tree in postfix order. A postfix traversal visits the right subtree first, then the left subtree, and finally the root.

   For this problem, binary tree should be represented as follows. The empty list is a binary tree with no elements, and a list of the form (*label left right*), is also a binary tree, where *label* is the atom that labels the tree's root, and *left* and *right* are binary trees. For example `(postorder '(y (r () (a () ())) (g () ())))` would return the list `(g a r y)`.

3. (5 points) Do exercise 10-6 on page 379 in the text.

4. (5 points) Do exercise 10-14 on page 383 in the text.

[1]See the instructor if you want to use a LISP on a home computer.

5. (5 points) Do exercise 10-15 on page 384 in the text.
6. (10 points) Do exercise 10-19 on page 385 in the text.
7. (10 points) Do exercise 10-21 on page 388 in the text.
8. (10 points) Do exercise 10-24 on page 391 in the text.
9. (10 points) Do exercise 10-28 on page 391 in the text. (Assume that the two arguments have the same shape.)
10. (10 points) Write a parameterless function `statically-scoped` that returns `t` if it is executed in a version of LISP with static scoping and `nil` if it is executed in a version of LISP with dynamic scoping.
11. (18 points) Discuss the costs and benefits of programs that make heavy use of functionals. Compare the costs and benefits of such programs with (i) LISP programs that do not use functionals and (ii) Ada programs written in an imperative style. Be specific and give examples to illustrate your points. Do not write more than about two pages (but don't feel that you must write two pages).

    Three of the points for this problem will go towards how well your argument is presented. For these points, we will consider such factors as whether you: present your points clearly, argue a praticular point of view instead of being wishy-washy, avoid being wordy, and have well constructed sentences and paragraphs.