Homework Assignment 4

Posted: 3/31/2010
Due: 4/13/2010

The assignment is due in class; please follow the instructions on the homework submission form.

You are expected to study the Haskell resources on the course homepage—a variety of different tutorials are listed; find one that works for you. Make sure to start early on the assignment so that you can raise any questions that might arise during office hours.

Use the provided skeleton files as a starting point.

Objectives

‣ Write explicitly recursive functions in Haskell.
‣ Implement functions using list operators in Haskell.
‣ Implement infinite lists in Haskell.
‣ Implement a simple composite data structure and a type class instance in Haskell.
‣ Iteratively find a solution to recursively-defined sets in Haskell (extra credit).

Related Files

Homework submission form:

Skeleton Haskell files:

Part 1

Implement the following functions using explicit tail-recursion:
‣ myLength to map a list of Int to its length;
‣ myMax to map a non-empty list of Int to the maximum element in the list;
‣ myMin to map a non-empty list of Int to the minimum element in the list;
‣ mySum to map a list of Int to the sum of its elements; and
‣ myMean to map a list of Int to the average of its elements.

Your solution should use neither pre-defined higher-order-functions such as map, foldl, or foldr, nor length, maximum, max, min, minimum, or sum. You may define and use your own higher-order-functions should you choose to do so.

Part 2

Implement each of the functions in Part 1 using map and foldl without resorting to explicit recursion. Do not use the built-in definitions of length, maximum, max, min, minimum, or sum. Instead, provide your own definitions as required.

Part 3

Implement the Fibonacci sequence as an infinite list of Integers.
Implement Recaman’s sequence as an infinite list of Integers.

Part 4

Implement an algebraic type Coord that implements simple 2D coordinates and accompanying arithmetic. In particular, the type should have three constructors Position (representing a 2D coordinate, encoded as a pair (x, y)), Vector (representing a distance, encoded as a pair (Δx, Δy)), and Scalar (representing a number). Make Coord a member of the built-in type class Num by implementing the following operations:

- adding a Vector (a, b) to a Position (x, y) yields a new Position (x+a, b+y);
- adding a Vector (a, b) to a Vector (c, d) yields a new Vector (a+c, b+d);
- subtracting a Vector from a Position yields a new position (in the inverse direction);
- subtracting a Position (x1, y1) from a Position (x2, y2) position yields a Vector (x2-x1, y2-y1);
- multiplying a Vector by a Scalar linearly scales the length of the vector, e.g., multiplying the vector (3, 3) by the scalar 5/3 should yield the vector (5, 5);
- multiplying a Position by a Position yields the dot product;
- negating a Position or Vector flips each component's sign;
- the absolute value of a Position is defined to be its distance from the origin as a Scalar;
- the absolute value of a Vector is its length.

Operations on Scalars (addition, subtraction, multiplication, etc.) should work as in normal arithmetic. All other combinations and operations are undefined and should result in a runtime error (using the error function). Your implementations of addition and multiplication should be commutative, i.e., satisfy a + b = b + a.

Extra Credit

Implement the recursive computation of LL(1) predict sets based on the data structures defined in extra.hs (see related files above).

Note: in order to receive partial credit, your solution has to be mostly working!

Guidelines

Your solution must be executable with ghci on the class host stetson.cs.unc.edu.

You may discuss possible approaches to Parts 1-4 with other students, but you cannot share source code. You cannot discuss the extra credit problem.

To receive full credit, you must explicitly declare the type of all top-level functions.

Please comment your code.

Deliverables

The Haskell source code for Parts 1–4 (and optionally the answer to the extra credit problem).

Grading

Your solution will be predominantly graded on correctness.

15 points — Part 1.
20 points — Part 2.
30 points — Part 3.
35 points — Part 4.

60 points — Extra Credit.