CPS506 Lab 4

Trickier Recursion, Nested Recursion

Preamble

In this lab you'll tackle some trickier recursive problems, some of which include nested recursion over lists of lists.

Lab Testing and Code Structure

Provided with this lab description is an Elixir mix project containing several different folders and files. In the **lib** directory is a file called **lab4.ex** which contains the module Lab4. You will add your functions to this module. In the folder **test** is a file called **lab4_test.exs**. This file contains several test cases for each function. Feel free to add your own tests if you wish. You can run these test cases on your code using the 'mix test' command from the top-level lab4 directory. Your code must compile *without warnings*. **FIX THOSE WARNINGS!**

The 'mix test' command can be run from your own terminal if you installed Elixir locally. This is just like the previous lab.

Lab Description

Five functions are described below. Solve each of these in provided mix project, just like you did in the previous Elixir lab. Also like the previous lab, you should use tail recursion.

1) **`catNestedStrings/1 –`** This function accepts a list that can contain values of any type. You should return the concatenation of all the strings in the list. This question is the same as the previous lab, but this time there can be nested lists. For example, if the input is: [5, 3, 7, "cat", 9, ["dog", ["horse", -5]], "mouse"], you should still return "catdoghorsemouse"

2) **`filterNestedStrings/1 –`** This function accepts a list that can contain values of any type. Build and return a new list with all the strings removed. This question is the same as the previous lab, but this time there can be nested lists. For example, if the input is: [5, 3, 7, "cat", 9, ["dog", ["horse", -5]], "mouse"], you should return [5, 3, 7, 9 [[-5]]]

3) **`tailFib/1 –`** This function accepts an integer argument, **n**, and returns the **nth** Fibonacci number. Assume the first two Fibonacci numbers are 1 and 1. That is, tailFib(1) == 1, tailFib(2) == 1. There is no tailFib(0). Your tail-recursive implementation must avoid the double recursive call. No $O(2^n)$!

4) **`giveChange/2` –** This function accepts an amount of money, and a list of coin values (in descending order). The function should return a list containing the minimum number of coins necessary to represent the money amount. The returned list should be in descending order. If it is not possible to make change for the amount with the provided list of coins, you should return the **`:error`** atom. Several examples are below.

amount:	**coins:**	**expected result:**
64	[50, 25, 10, 5, 1]	[50, 10, 1, 1, 1, 1]
100	[42, 17, 11, 6, 1]	[42, 42, 11, 1, 1, 1, 1, 1]
64	[50, 25, 10, 5]	:error

You may assume the coin values are provided such that the greedy solution will always be correct. Subtract the largest coin possible at each step.

5) **`reduce/3` –** Consider MyEnum.reduce from the Elixir lecture slides lecture. Lab4.reduce will build on this. You will add an implementation that handles the optional 3rd argument to initialize acc.

For example – the following two examples, exactly as written, should work correctly:

```
Lab4.reduce([1, 2, 3], fn(x, acc) -> x+acc end)
Lab4.reduce([1, 2, 3], 10, fn(x, acc) -> x+acc end)
```

You may use helper functions if you wish, so long as the user can invoke your function from the outside world as seen above.

Submission

Labs are to be completed and submitted *individually*. Submit your **lab4.ex** file containing the Lab4 module and all completed functions on D2L, under the submission for Lab4. You do not need to submit the tester files.