Programming languages (TC-2006)

Homework 14

In this homework, you will explore a good practice for recursion: tail-recursive functions. Although you will practice using Erlang, this concept can be applied to any functional programming language. Since the main purpose of this homework is to introduce the concept of tail recursion, any solution that does not implement tail recursion will not be considered for grading. For example, none of these problems can be solved by using higher-order functions.

Let f be a function which, in its body, contains a call to a function g (different from f or equal to f). The call of g is said to be a tail-call if the function f returns the value returned by g without having to perform any other computation. We say that the function f is tail-recursive if all the recursive calls present in f are tail calls.

For example, analyze the following function that calculates the factorial of a number:

```
factorial(0) -> 1;
factorial(N) when N > 0 -> N * factorial(N - 1).
```

This function is not tail-recursive because the value returned by `factorial(N - 1)` is used in `factorial(N)` and call to `factorial(N - 1)` is not the last thing done by `factorial(N)`.

A tail-recursive approach to calculate the factorial of a number is as follows:

```
factorial(N) -> factorialAux(N, 1).

factorialAux (N, R) -> if
      N =< 1 -> R;
      true -> factorialAux(N - 1, N * R)
   end.
```

1 `pow` (20%)

Write a **tail-recursive function** in Erlang that calculates the power of a number. As a reference, use the following function (which is not tail-recursive):

```
pow(_, 0) -> 1;
pow(N, X) when X > 0 -> N * pow(N, X - 1).
```

2 `fibonacci` (20%)

The fibonacci numbers are the numbers in the following sequence and characterized by the fact that every number in it is the sum of the two preceding ones: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... Write a **tail-recursive function** in Erlang that calculates the f_i number in the sequence. As a reference, use the following function (which is not tail-recursive):

```
fibonacci(0) -> 0;
fibonacci(1) -> 1;
fibonacci(X) when X >= 1 -> fibonacci(X - 1) + fibonacci(X - 2).
```

3 `reverse` (20%)

Write a **tail-recursive function** in Erlang that reverses a list. As a reference, use the following function (which is not tail-recursive):

```
reverse([]) -> [];
reverse([H | T]) when is_list(H)-> reverse(T) ++ [reverse(H)];
reverse([H | T]) -> reverse(T) ++ [H].
```

4 `count` (20%)

Write a **tail-recursive function** in Erlang that counts the number of occurrences of an element in the list. As a reference, use the following function (which is not tail-recursive):

```
count([], _) -> 0;
count([H | T], X) -> if
      H == X -> 1 + count(T, X);
      true -> count(T, X)
   end.
```

5 `sum` (20%)

Write a **tail-recursive function** in Erlang that sums the elements in the list. As a reference, use the following function (which is not tail-recursive):

```
sum([]) -> 0;
sum([H | T]) -> H + sum(T).
```

Deliverables

Prepare an ERL file that contains the functions requested (in its corresponding module) and submit it to Canvas. **Please, do not submit other formats but ERL**. To prepare your ERL file, use the code template distributed along with this document. The template contains some test cases for each function to help you verify that your codes work as requested.

I promise to apply my knowledge, strive for its development, and not use unauthorized or illegal means to complete this activity, following the Tecnológico de Monterrey Student Code of Honor.