Functional and Logic Programming

Home Assignment 6 - Erlang

Due: Monday, 11.6.2019 - 23:55

Instructions

- Please create a source file called ***hw6.erl*** and put all the answers there.
  The file should start with a comment which contains **both** of your **full names** (in English) and **ID**, the **module** declaration with the name of the module (**hw6**), and the line "**-compile(export_all).**" which will export all of the functions in the code.

```
% Nir Koren
% 654321986

% Tom Barak
% 654321987
-module(hw6).
-compile(export_all).
```

- Make sure the file **is valid** by compiling it.
  A valid file will compile without any errors or warnings.
- Be sure to write functions with **exactly the specified name** for each exercise.
  You may create additional auxiliary/helper functions with whatever names you wish.
- Try to write **small functions** which perform just **a single task**, and then **combine** them to create more complex functions.

Exercises

1. Implement the function **reverse** which takes a list and reverses the order of its elements.
   Note: Your solution must be tail recursive.

   Examples:
   hw6: reverse ([]). **= []**
   hw6:reverse([1]). **= [1]**
   hw6:reverse([1,2,3,4]). **= [4,3,2,1]**

2. Implement the function **splitter** which takes a list of tuples and splits it into 2 lists (inside a tuple) where the elements of each list are the corresponding tuple elements from the original list.

   Examples:
   hw6: splitter ([]). **= {[],[]}**
   hw6: splitter ([{1,2}]). **= {[1],[2]}**
   hw6: splitter ([{1,2},{3,4},{5,6}]). **= {[1,3,5],[2,4,6]}**

3. Create a process that will wait in a loop for a message.
   Depending on the message, the process should either print the message or terminate.
   Your code should support the following interface:
   - start_server()
   - print(Msg)
   - stop_server()

   Your code should output the same as the examples.
   Hint: The **register** function can help.
   https://www.tutorialspoint.com/erlang/erlang_register.htm

   Examples:
```
> hw6:start_server().
ok
> hw6:print(hello).
hello
ok
> hw6:print(42).
42
ok
> hw6:print({hello,42}).
{hello,42}
ok
> hw6:stop_server().
stopped
```

4. To represent **binary trees** we'll use the **atoms** `leaf` and `node`.
Each tree is a **tuple** of one of two forms:

- `{leaf, X}` - represents a `leaf` which has the value `X`.
- `{L, node, R}` - represents a `node` which has the left child `L` and the right child `R`, where both `L` and `R` are also binary trees.

a) Implement the function `sumTree` which takes a tree and **sums** all the values in its leaves **sequentially**, within a single process.

b) Implement the function `sumTreeConc` which takes a tree and **sums** all the values in its leaves **concurrently**, by creating a new process for each node in the tree.
When the function is applied to the pattern `{L, node, R}` the function spawns a new process to compute the sum of `L`, and a new process to compute the sum of `R`, and waits for the results before adding them up.